Strategic Research Roadmap for European Web Security
FP7-ICT-2011.1.4, Project No. 318097
http://www.strews.eu/

Deliverable D3.2
European Web Security Roadmap

Deliverable details

Deliverable version: *v1.0*
Date of delivery: *31.10.2015*
Editors: *Martin Johns*

Classification: *public*
Due on: *M36*
Total pages: *203*

List of Contributors:
Lieven Desmet, Frank Piessens, Philippe De Ryck, Wouter Joosen, Rigo Wenning, Bert Bos, Stephen Farrell, Sebastian Lekies

Partners: ERCIM/W3C, SAP, TCD, KU Leuven

Executive Summary

In this document, we thoroughly assess the current state of web application security in respect to state-of-the-practice, state-of-the-art, research, and standardisation with special attention to the European aspect. The collected data is then utilized to successively define a near to mid-term research roadmap for Web security to guide impactful research and development of secure and trustworthy Web technologies. In this context the deliverable collects areas of Web security which are still underdeveloped, identifies missing pieces in the research landscape, and points out promising directions for future research. In addition connections between research and standardisation activities are explored, as well as existing mismatches in the area are shown. This way, the deliverable provides a bigger picture on the field of Web security research, to aid the decision making process, when it comes to create new research/standardisation actives and future research projects/work programs.

To structure this ambitious project, Part I first defines a systematic methodology for data collection and analysis. This methodology is based on five well defined objectives which were directly derived from the STREWS mission statement:

(OBJ.1) Identify significant gaps between the state-of-the-practice and current research results.

(OBJ.2) Identify mismatches in between ongoing/future standardisation and research activities in the field of Web security and the needs of the Web's practitioners.

(OBJ.3) Identify the emerging topics and future hot spots of Web security.

(OBJ.4) Map standardization and research efforts to the observed emerging topics in Web security. Identify topics that require further attention or are not covered yet.

(OBJ.5) Obtain current information on the state of European research in the field of Web security.

Guided by these objectives, suitable data sources were identified, which in total span the full spectrum of web application security. These data sources were chosen according to their capability to cover the four STREWS focus areas (*state-of-the-practice*, *research and innovation*, *standardisation activities*, and *emerging topics*). For each of the chosen data sources a clear data collection and analysis process was determined, to ensure that the resulting outcome helps to achieve the deliverable's objectives.

After the selection process a total of ten distinct data sources were selected to appear in this document:

(DS.1) The State-of-the-Practice in today's Web Software

(DS.2) Selected Empirical Studies

(DS.3) Observable Gaps between the State-of-the-Art and the State-of-the-Practice

(DS.4) Interactive Survey

(DS.5) Review of related NoE and Policy activities

(DS.6) STREWS Workshops

(DS.7) Standardisation Activities

(DS.8) STREWS Case Study 1: WebRTC

(DS.9) STREWS Case Study 2: Web Security Architectures

(DS.10) Cybersecurity

Each of these distinct topics is explored in depth in Part II of the deliverable. The sum of the collected reports provides a comprehensive overview on the current state of web application security and web application security research.

In the second half of Part I, the collected data is analysed according to the goals of the deliverable. For this, we first draw individual conclusions from the distinct data sources. Then, in a second step, these results are correlated on an objective level. In this step, we identify and explore the emerging and hot topics in web security that require future attention from research, practice and standardisation, namely: *client-side complexity*, *JavaScript sandboxing*, *server-driven security policies*, *JavaScript crypto and hardware tokens*, *the end of the client-server paradigm*, *web privacy*, and *advancing web authentication and session tracking*.

Finally, we use the collected insight to pinpoint the upcoming security research challenges for the European web, which either directly result from the emerging topics or materialized as notable insights from the objective-level data correlation:

- Challenge 1: *Revisiting classic attacks*
- Challenge 2: *Handling the extending web paradigm*
- Challenge 3: *Realizing real end-to-end security*
- Challenge 4: *Increasing End-user Security and Privacy*

The combination of the identified emerging topics and the overarching research challenges results in an exiting and promising research roadmap for the mid to long term. We expect, that following this roadmap will lead to impactful results, which address the future security problems of the Web, while being well suited to be adopted by practitioners and standardisation.

Contents

List of Figures

Chapter 1

Introduction

This deliverable presents the final results of the of the STREWS research roadmapping activity.

As detailed in the STREWS description of work, the roadmap preparation process was structured in two overlapping phases: Data collection and Data analysis.

Given this two step process with significant emphasis on the data collection part, it is apparent that the roadmapping activity is intended to be heavily data driven, i.e., the roadmap's final results are required to be based on solid evidence provided by a comprehensive data basis. Thus, according to this strategy this roadmap deliverable is organised in two distinct parts:

In the first chapters of Part I, we present the chosen methodology, that was be applied thorough the roadmapping activity's data collection and analysis process. The roadmap's scope, focus areas and objectives are be specified and a set of data sources, that are suited to reach these objectives are presented. In this context, we show how the STREWS deliverables and activities were utilized for the roadmapping preparation task. Based on the collected data, we draw individual and overarching conclusions Part I.

In Part II, we present detailed information on the chosen data sources. For each source, we present collected data and discuss the data collection/analysis process.

Roadmap preparation phases

As mentioned above, the preparation process of this document was structured in two phases. In this section, we give a brief overview on these phase's goals, which determined the approach in which the preparation was executed.

A. Data collection: In the STREWS description of work, the roadmap's underlying data and the approach to collect, create and evaluate the data was characterized as follows:

1. Gathering of information on existing or emerging European (and international) research activities, results, groups, and projects.
2. Ongoing evaluation of the input provided by the STREWS case studies and community events.
3. Systematic identification of ongoing and planned standardisation efforts of the W3C and IETF working groups.
4. Matching of results from steps 1. to 3. to create an comprehensive overview on the current Web security research landscape and how the current research and standardisation activities align.

Using this guidance, we derived a set of focus areas and objectives for the roadmap (see Section 2.2, which in turn aided the identification of suitable data sources (see Section 2.3). For

these data sources, we define a high level data collection approach and specify the data's role in the analysis process.

B. Data analysis: Then, in the latter chapters of Part I, we present the conclusions that can be directly drawn from the collected data. Wherever applicable, special focus of this analysis will be on:

- Emerging research aspects that are currently not sufficiently represented in the existing research landscape.
- Potential synergies between research and standardisation, which are not fully realised yet.
- Mismatches between research and standardisation, i.e., research efforts and results without counterparts in standardisation and standardisation efforts which are not sufficiently supported by corresponding research.
- Mismatch between existing problems of practitioners and activities in standardisation/research.
- Unsolved problems, both classic and emerging in the area of web application security.

For this, in Chapter 4 we first draw individual conclusions from the individual data sources and then correlate the gained insight according to the roadmaps objectives (see Sec. 2.2). Based on this objective-driven reasoning on the collected data, Part I concludes in Section 6 with a set of overarching conclusions and a list of upcoming challenges that together should shape the future of European research on web application security.

Part I

European Web Security Roadmap

FP7-ICT-2011.1.4
Project No. 318097

Chapter 2

Roadmap Preparation Methodology

2.1 Overview

This chapter documents the chosen approach towards roadmap preparation. As discussed before, the chosen methodology is highly data-driven. This means, based on collected data from relevant sources, a comprehensive overview on the state of web security in all areas with significance for the roadmap will be compiled.

As symbolised in Figure 2.1, STREWS is primarily focused on the following four areas:

- State-of-Practice as experienced and executed by practitioners of the (EU) software industry, to obtain information how Web security is currently handled in the productive software engineering world.
- Research and innovation conducted in ongoing EU-funded research projects, as a measure on the current activities and awareness in the European research community.
- Standardization activities, which provide valuable pointers towards the future of the Web platform.
- Emerging topics, to capture the upcoming hotspots of Web security.

The collected data allows to match the achieved progress and observed shortcoming in the focus areas, which in turn leads to conclusions with respect to maturity of results, uptake of research in standardisation and practice, mismatching agendas, unsolved problems and potential future directions.

2.2 Objectives

More precisely the roadmapping deliverable aims to provide the basis to reach the following objectives:

(OBJ.1) Identify significant gaps between the state-of-the-practice and current research results.

(OBJ.2) Identify mismatches in between ongoing/future standardisation and research activities in the field of Web security and the needs of the Web's practitioners.

(OBJ.3) Identify the emerging topics and future hot spots of Web security.

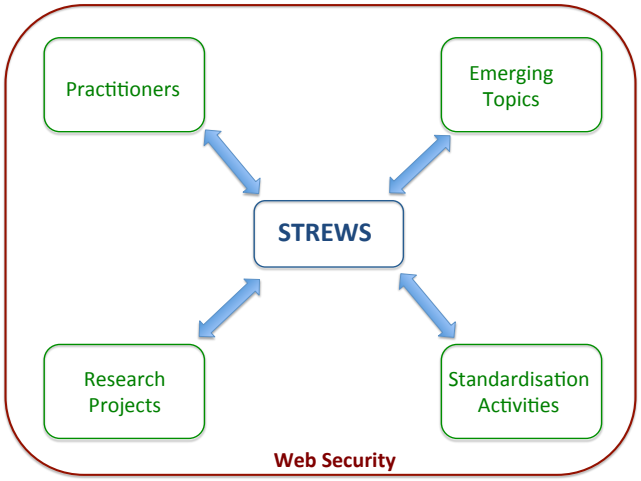


Figure 2.1: Focus areas of the STREWS data collection approach

(OBJ.4) Map standardization and research efforts to the observed emerging topics in Web security. Identify topics that require further attention or are not covered yet.

(OBJ.5) Obtain current information on the state of European research in the field of Web security.

The combined insight gained through following these objectives allows a neutral evaluation, where (European) research and practice in the field of Web security currently stands and in which direction the field and community has to progress to ensure long-term security for (European) Web applications and their users.

For instance Objective (OBJ.1) enables us to pinpoint areas, in which we still encounter security problems in real-world systems, for which the underlying problem is considered to be "solved" by the research community. A motivational example for such a mismatch is SQL Injection, which is received considerable attention in the middle of the last decade [90, 194], which resulted in various potential solutions covering a wide range of scenarios. Nonetheless, according to evidence collected through publicly available vulnerability and incident disclosures, SQL Injection is still a highly wide spread problem that can be frequently encountered in current productive (Web) software.

2.3 Approach

In order to achieve the objectives given in Section 2.2, we follow a systematic methodology, which will be presented in the remainder of this section.

As motivated above, the roadmap is centered around four focus areas: State-of-the-practice, Research and Innovation, Standardisation, and Emerging Topics. Hence, as a first step, for each of these areas a set of matching data sources was identified. Selection criteria was, that these data sources should be sufficient to provide comprehensive insight on the current state of the focus area and its potential future.

In a next step, the identified potential data sources are mapped to the objectives of Section 2.2. This mapping ensures that all for all objectives sufficient data is available in the course of roadmap preparation.

Then, for each chosen data source, fitting techniques for data collection were found and prepared. Given the nature of the STREWS project, factors which steer the decisions in this step include:

- *Effort and feasibility*: It had to be feasible to execute the envisioned data collection methodology successfully with the resources available in the project and in the timespan of the project's lifetime. This especially has to be taken into account for data collection activities that either necessitate a substantial amount of manual effort or require the cooperation of persons/entities outside of the project's partners and their organisations.
- *Expected expressiveness*: Each selected data source should be capable to provide meaningful data. More precisely, the obtained data should either be of a broad enough nature, to represent a comprehensive overview of a certain (sub) topic within the focus areas, or be a promising target for a topical deep, in which we conduct detailed explorations of selected matters.
- *Timeliness:* It had to be feasible to obtain the targeted data under the time restrictions present in the project. This is especially significant for data that covers longer time periods. While feasible for already existing data sources (such as public vulnerability databases), this requirement can be a limiting factor for self-collected data. Hence, all empirical studies, that are STREWS driven need to be designed in a fashion that the gained information is valuable as a snapshot of the current state, opposed to a recording of long-time trends.

Based on these criteria, the STREWS consortium decided on a set of ten individual data sources, that provided the data basis for the roadmapping activity. The rationale for choosing the data sources will be given in Section 3.3 and the collected data, together with precise information on the applied data gathering techniques, is presented in Part II of this document.

To ensure that scope of the identified data sources are sufficient to provide comprehensive insight, it was essential to verify that envisioned collected data covers the focus areas so that Section 2.2's objectives could be reached. To do so, we provide a mapping between data sources and focus areas / objectives in Section 3.3.

Chapter 3

Data Collection Process

3.1 Overview on targeted data sources

Based on the objectives given in Section 2.2 and the selection criteria of Section 2.3, a set of various data sources or respectively data collection approaches (see Section 3.2 for details) were selected. Theses targeted data sources can be divided into five distinct classes:

- *State-of-the-practice*:

  To assess the past achievements, current problems, and future challenges of Web security research, one first needs a firm grasp on the actual state of affairs in the field of productive (web) software design and system operation. Hence, it is essential to survey public data sources that record and report security vulnerabilities and incidents. Especially interesting are observable mismatches between research and practice, i.e., cases in which a underlying security problem is considered to be "solved", while public incident data clearly shows that real-world productive software still suffers from the corresponding class of security vulnerabilities.

  Furthermore, in this context, selective deep dives into significant sub topics via large scale studies can shed light on recent non-trivial developments.

- *Research in European projects*:

  To efficiently capture data concerning research on Web security that has potential impact for a European roadmap on Web security research, it is essential to capture both current related research efforts in European research projects as well as the current approach towards Web security in non-Web security EU projects.

- *Community input:*

  STREWS does not exist in a vacuum. Instead, the field of Web applications is filled with a multitude of researchers (both academic as well as industry), practitioners (ranging from small scale start-ups to Fortune 500 companies) and other stakeholders. Hence, explicit external input from the Web community is essential.

- *Related activities in standardisation*:

  Standardisation is one of the four focus areas of the roadmap. Thus, one of the pillars of the roadmap's data basis consists of a comprehensive overview on current and upcoming standardisation efforts with relations to Web security and security sensitive Web technologies.

- *STREWS case studies*:

The STREWS consortium is composed out of the members of the two most relevant standardisation bodies (W3C and IETF), one of Europe's leading academic Web security research groups (KU Leuven), and Europe's largest software vendor (SAP). Thus, when it comes to identifying emerging topics which touch the project's focus areas, the project partners are in expert roles. Therefore, the two STREWS case are designated providers of guiding information for the research roadmap. A special focus of the case studies is the examination of the interplay of research and standardisation in the context of emerging topics, a subject for which only few (if any) alternative data sources exist.

In the following section, we will provide information of the individual data sources, organised according to the classification scheme given above.

3.2 Data collection approach

As outline in the previous section, we identified a set of high-level area for which the roadmaps underlying data is drawn. In this section, we specify these areas' concrete data sources, that were chosen to be investigated. In the context of this section, we concentrate on the reasoning, why these specific data sources are good candidates for the process. Concrete information on the applied data collecting methodology and results can be found in the corresponding chapters of Part II in this document.

The individual data sources will be enumerated using identifiers of the form *(DS.x)*, where x will be the data sources ID. This will allow a more compact assessment of the approach's coverage in Section 3.3.

3.2.1 The State-of-the Practice

Three distinct data sources were chosen:

(DS.1) Public vulnerability/incident reports

Several companies and organisations collect statistical data about security incidents and vulnerabilities in real-world software, products and projects. As many of these data collection endeavors have been active for multiple years in the past, they offer valuable insight in the development of the vulnerability/incident landscape and provide insight in the current trends and developments in this area. A special focus during data processing will be

- on the development of Web related security topics compared to general security problems and
- on observable gaps between the recorded security problems in real-world software and the perceived state-of-the-art in software development and security research.

To obtain a comprehensive and expressive data set, a selection of four external data survey efforts has been made (Cenzic's Application Vulnerability Trends Report, the Common Vulnerabilities and Exposures Database, Trustwave's Global Security Reports, and WhiteHat Security's Website Security Statistics Reports). Depending on the specifics of the respective data source, either the compiled data is post-processed to obtain the desired insight or (as it is for example the case with the CVE data), specific technical procedures have been implemented to compile the targeted statistics.

(DS.2) Empirical studies

While general statistics compiled by third parties are a valuable asset in respect to assessing the overall state of the subject, they do not suffice when it comes to examine more specific aspect of

current practices Web security and topical deep dives. For this reason, the STREWS consortium decided to conduct a limited set of empirical studies to explore selected topics.

For this iteration of the deliverable, we report on three large-scale empirical studies:

- *Assessment of the observable security practices of the European Web:* In the recent years various security mechanism have been added to the Web, some in the form of HTTP response headers, some in the form of other observable protocol artifacts, and some in the form of coding conventions. Wether these techniques are picked up by a given Web application/site can be determined by analysing the site's HTTP responses and the corresponding HTML. In a large scale examination of popular European Web sites, we explored the current level of awareness and adoption of these security technologies in the European Web.
- *Large-scale analysis on client-side complexity based on DOM-based XSS measurements:* The recent years have brought a significant push of application logic from the server to the client-side. Modern Web applications consist of a considerable amount of JavaScript code that is executed in the user's browser. Only little inside exists on the complexity of this code and the security implications, that are connected with this shift. Hence, as a first measure we conducted a practical study on DOM-based Cross-site Scripting, a sub-class of XSS (see Sec. 7.1.1) that is caused by insecure JavaScript code.
- *Large-scale Analysis of Third-party Security Seals* A third-party security seal is typically an image that can be embedded in a website to signal consumers that the website has been scanned by a security company and has been found to be without security issues. We have performed a large-scale analysis of third-party security seal providers, to assess whether these seals can be trusted and to what extent the seal providers can accurate detect vulnerabilities in a given websites. Moreover, we investigated the existing ecosystem of third-party security seals.

(DS.3) Observable gaps between the State-of-the-Practice and the State-of-the-Art

The STREWS "Web-platform security guide: Security assessment of the Web ecosystem" (see STREWS Deliverable D1.1 [64]) provided a comprehensive overview on today's security challenges of the Web. An important part of the report's content was, for all classes of security problems that were discussed in the report, to systematically document both the current state-of-the-art, in respect to handling the security issues, as well as the perceivable state-of-the-practice, as it can be obtained from external data. In this deliverable, we revisit the lessons learned from D1.1 in the context of the roadmap preparation task and examine the observable gaps between the State-of-the-Practice and the State-of-the-Art that were found during preparation of the report.

3.2.2 Research in European projects & Community Input

As motivated above, European research projects are a valuable source for insight in the current awareness and practices of Web security in the European research community. For this reason, two methods for gathering relevant information have been identified:

(DS.4) Interactive survey

For one, we conducted an interactive survey primarily targeted at European research projects, in which we gathered data on awareness and application of Web technologies and their corresponding security aspects.

Furthermore, to enable additional insight, the survey was designed in a fashion that the survey's questions can be also answered outside of the context of a EU project. This in turn allowed us to distribute the questionnaire widely.

(DS.5) Review of related NoE and policy activities

To learn how the (Web) security topic has been handled by previous projects and activities, a comprehensive examination of relevant NoE and policy activities will be conducted.

(DS.6) STREWS Workshops

STREWS organised a set of workshops, in which we both disseminated the project's results and collected input from the Web, standards, security and research community in respect to current and upcoming challenges in Web security.

3.2.3 Standardisation

As outlined in Section 2.1, standardisation efforts on Web security topics or affecting technical aspect of the Web that have a direct connection to security is one of the four focus areas of STREWS. Hence, for the roadmap's data collection activity, it was essential to both compile a comprehensive overview of ongoing standardisation activities, as well as get an impression on the respective standardisation organisation's process and potential roadmap for the near future and mid term.

(DS.7) Related activities at IETF, W3C and other SDOs

According to the data collection goals motivated above, a deep dive into the workings of the W3C and IETF have been conducted, accompanied with an overview on related activities at IEEE, MAAWG, ETSI, and OASIS.

(DS.10) Cybersecurity Initiatives

Through the Dissemination activities and community building exercises, STREWS has gathered feedback on the Cybersecurity activities of the European Commission including an analysis of the underlying legal framework. This was facilitated by W3C and IETF participation in those Initiatives and can thus be classified as input from standardisation

3.2.4 STREWS Case Studies

As motivated in Section 3.1, the STREWS case studies are a powerful means to exploit the strength of the STREWS consortium – the combination of multiple approaches and viewpoint towards (research / standardisation / industry) on the emerging topics of Web security.

(DS.8) WebRTC case study

The first STREWS case study was on the topic of WebRTC – real-time peer-to-peer communication in the Web browser. The study has concluded and its results we reported in detail in STREWS deliverable D1.2. The key results will be reiterated in this document, to highlight the learned lessons.

(DS.9) Web Security Architectures case study

The second STREWS case study explored the field of Web security architectures, as there is a clear need in the web community to build more sophisticated encapsulation and method invocation mechanisms.

	State-of-the-Practice	Research	Standardization	Emerging Topics
(DS.1)	X			
(DS.2)	X	X	(X)	X
(DS.3)	X	X	X	X
(DS.4)	(X)	X	(X)	(X)
(DS.5)		X		
(DS.6)	X	X	X	X
(DS.7)			X	
(DS.8)		X	X	X
(DS.9)	(X)	X	X	X
(DS.10)	X		X	

Table 3.1: Mapping data sources to focus areas

3.3 Data-analysis methodology

To be able to base the roadmap on a sound foundation, was essential to verify, that the chosen data sources fulfill two criteria: For one, the data sources had to comprehensively cover the four STREWS focus areas (*State-of-the-Practice, Emerging Topics, Research, Standardization*). Equally important, the collected data had to be sufficient to achieve the objectives discussed in Section 2.2.

In this section, we conduct this verification step and outline how the data sources were targeted to be used in the roadmap's creation process (please refer to Tables 3.2 and 3.1 for a quick overview on matching the data sources with the roadmap's focus areas and objectives):

(DS.1) Public vulnerability/incident reports

Coverage of focus areas: This data source mainly provided insight corresponding to the *State-of-the-Practice*, as it reflects the most pressing, currently existing challenges in real-world Web security.

Objectives: The data source fed into (OBJ.1). The external statistics report on security incidents in productive (Web) software. Comparing these results to information on the state of the art in research collected via (DS.2), (DS.3), (DS.4) and the STREWS deliverable D1.1 allowed to pinpoint significant gaps between the state-of-the-practice and current research results.

(DS.2) Empirical studies

Coverage of focus areas: Due to the selected topics of the three conducted studies, all four focus areas are covered: The first study (Assessment of the observable security practices of the European Web) specifically examined the uptake of freshly standardised security mechanisms by praticioners, thus, providing insight into the areas *State-of-the-Practice* and *Standardization*, while in the second empirical study the attention was on the *Emerging Topic* of DOM-based XSS both from the *Research* and the *State-of-the-Practice* perspective.

Objectives: The first empirical study helped in following (OBJ.2), via examining the uptake of security mechanisms (especially under consideration of the results from (DS.1) by practitioners. The second empirical study was mainly concerned with (OBJ.3) through its examination of emerging security problems that result from the ongoing shift towards client-side complexity. It also helped towards (OBJ.1), as the underlying problem of XSS has received considerable attention from the research community. The study provided insight, if these research results had impact on the client-side facet of XSS.

	(OBJ.1)	(OBJ.2)	(OBJ.3)	(OBJ.4)	(OBJ.5)
(DS.1)	X				
(DS.2)	X	X	X		
(DS.3)	X	X			
(DS.4)	X	X	X	X	X
(DS.5)					X
(DS.6)	(X)		X	(X)	(X)
(DS.7)	X			X	
(DS.8)		X	X	X	
(DS.9)		X		X	X
(DS.10)	X			X	

Table 3.2: Mapping data sources to objectives

(DS.3) Observable gaps between the State-of-the-Practice and the State-of-the-Art

Coverage of focus areas: (DS.3)'s underlying data was provided by the STREWS deliverable D1.1 ("Web-platform security guide: Security assessment of the Web ecosystem"), which was designed to provide a first foundational overview of the four STREWS focus areas, to base further project activities upon. Thus, all areas are covered.

Objectives: While the contents of the Web-platform security guide cover a lot of ground, in the context of the roadmap preparation task, we mainly extracted information in respect to objective (OBJ.1) from its results. For all covered aspects of the document, both the current state-of-the-art in respect to research as well as the current state-of-the-pratice, as observable in the wild, are documented. Thus, via an additional analysis step, the gaps between these two views in to the problems could be apprehended.

(DS.4) Interactive survey

Coverage of focus areas: The primary target of the survey were European research projects, with a potential further outreach towards a broader survey group. Thus, a main focal point was on the *Research* area. The contents of the survey was designed to touch all four focus areas of STREWS.

Objectives: The design of the survey's questionnaire targeted to cover all roadmap preparation objectives via matching questions. For instance, we queried what the survey partitioners consider to be the emerging hot spots of Web security, feeding into objective (OBJ.3).

(DS.5) Review of related NoE and policy activities

Coverage of focus areas: This topics was clearly focussed on the *Research* area, as the underlying data basis consists of published documents of European research projects.

Objectives: The underlying examination of the reports was targeted to help identify the current state of European Web security research, thus, provided data for objective (OBJ.5).

(DS.6) STREWS Workshops

Coverage of focus areas: The STREWS workshops were targeted to attract attendees from all communities relevant to STREWS, namely the *Research*, *Standardiation*, and *Practitioners* community. Furthermore, the motivating topics for the workshops were be drawn out of the set

of *Emerging Topics* (e.g., pervasive monitoring in the case of the STRINT workshop) and seeked input from the communities specifically on such topics.

Objectives: The workshops were a tool to solicit input from outside of the project from the relevant communities, in the form of workshop contributions from the attendees. Depending on the topic of the respective workshop this input contributed to different objectives. A predominant motivation was to achieve isight need for (OBJ.3) (identification of hot and emerging topics).

(DS.7) Related activities at IETF, W3C and other SDOs

Coverage of focus areas: This data source was mainly concerned with collecting relevant information concerning *Standardization*.

Objectives: The data collection effort was targeted to help with the objectives (OBJ.2) and (OBJ.4), through providing information on the current and future direction of ongoing standardization which touches Web security. This allowed in a secondary step through correlating the collected information with the results concerning the areas *State-of-the-Practice* (e.g., via (DS.1), (DS.2), and (DS.3)), to find mismatches with the needs of the practicioners, and how the *emerging topics* (via (DS.2), (DS.6), (DS.8) and (DS.9)) align with the near to mid term future of standardization.

(DS.8) WebRTC case study

Coverage of focus areas: WebRTC is an *Emerging Topic* that receives a lot of attention from both *Standardization* and *Research*, while being still to novel to be fully embraced by the *Practitioners*.

Objectives: WebRTC was an outstanding opportunity to observe the standardisation of an emerging topic with high security impact is realised. This provided valuable insight for objectives (OBJ.3) as well as (OBJ.4). Based on our independent security evaluation of the technology, in which the viewpoint of endusers and developers was taken, a cross check if the quality and coverage of the currently discussed specification and implementation is sufficient for practitioners to embrace WebRTC (feeding into Objective (OBJ.2)). Especially interesting in this context was the examination if the involved standardisation processes (spread over multiple standardisation organisations) was capable to handle the full attack surface of such a complex endeavor such as WebRTC.

(DS.9) Web security architectures case study

Coverage of focus areas: The topic of the case study is highly motivated by the actual needs of the *practitioners*, constantly challenged by *emerging technological additions* to the Web platform and is receiving considerable attention by both *researchers* as well as from the *standardisation* community, resulting in excellent scoping for the focus areas of STREWS.

Objectives: Compared to the WebRTC case study, the second STREWS case study was conceptionalised to be of broader reach. The field of security architectures addresses security problems that are as old as the Web. Also while WebRTC is strongly driven by standardisation, for Web security architectures results from research are a significant steering source, giving evidence for objective (OBJ.5). Furthermore, through careful crosschecking with the results from (DS.1), (DS.2), and (DS.3), this data collecting exercise was utilized in the context of the objectives (OBJ.2) and (OBJ.4).

(DS.10) Cyber Security Initiatives

Coverage of focus areas: The NIS Directive but also the eIDAS Regulation are on the top of the Agenda of the EU Digital Single Market strategy. Most of the Digital Single Market is dependent on Web technologies.

Objectives: The Cybersecurity area is very active and very prominent in the political area. It is not as prominent in research and standardisation where the movements are accommodated with some scepticism to use a mild wording. STREWS gathered the new requirements from those Cybersecurity Initiatives and matched them against the evolution in Web security to find gaps. This allows the roadmap to suggest activities to close the identified rather large gap between the regulatory positivism and the world as it is implemented and laid down in Specifications.

3.4 Analysis process

In the previous section, the decision process behind the data source selection has been documented, accompanied with an explicit listing of the chosen sources and their respective data gathering and analysis methodologies. As discussed, it can be seen in Section 3.3 and Tables 3.1 & 3.2, the selected approach is well suited to cover all STREWS focus areas and resulted in well suited input towards reaching the roadmap's objectives.

Based on the collected data, we conducted the following steps:

- *Data analysis:* For the collected data, the first step is a data source specific analysis step, in which the information is pre-processed under the context of the roadmap's objectives to prepare the data correlation. The results from this step will be presented in Chapter 4.
- *Correlating separate data sources:* Then in Chapter 5, an objective-level data correlation sets the various information sources into relation, to detect overarching trends or mismatching results on the same subject, depending on the various angles.
- *Identification of upcoming challenges:* Finally, using the results of these activities, we draw our conclusions in respect to the current state and near/midterm future of Web security in Chapter 6. This enables us to highlight problem domains, that do not receive enough attention, mismatches between the views/activities in the various STREWS focusareas, and emerging topics, that need to be addressed in the future.

Chapter 4

Assessing the State of Web Security

In this chapter we highlight selected key takeaways, that result from the roadmap's data sources. However, given the amount of compiled data and analysis on the various data sources, this chapter's provided information is to be regarded as only a brief insight into a bigger picture.

Thus, for readers of this roadmap document, it is highly recommended to resort to the full reports in the data sources' respective chapters, contained in Part II of this document, in case a given topic is of interest. Furthermore, for information contained in the STREWS case study data sources (DS.8) and (DS.9), the corresponding STREWS deliverables with the full case study reports are suggested reading material.

4.1 External security statistics (DS.1)

In Chapter 7, to obtain a realistic insight in the current state of web software and applications, we provide a comprehensive overview of several externally compiled statistics on web security, such as the *Cenzic Application Vulnerability Trends Report*, the *Common Vulnerabilities and Exposures (CVE) index* or the *White Hat Security Website Security Statistics Reports*.

Using these sources, it has to be observed, that Web related security issues are a major problem for productive, real-world software. Even after a considerable amount of years in which practitioners, researchers, browser vendors and the standardisation community have put work into addressing the underlying problems and developing defenses, Web vulnerabilities plaque a high number of applications.

The numbers of vulnerable websites, a number between 77% and 96%, is startling, especially, because well-known vulnerabilities such as SQL injection are still ranked on a high position. The number of web attacks in general fluctuated over the past years and, although the number of some kinds of attacks decreased over the time, we cannot see an obvious improvement. When some kinds of attacks are falling, other ones arises and attacks we thought of being defeated in one year, celebrated their comeback in the following.

Hence, we are far away from a secure Web.

4.2 Empirical studies (DS.2)

4.2.1 Large-scale Security Analysis of the European Web

Dataset 1: September 2013

In Section 8.1, we report on the state of security for more than 22,000 websites that originate in 28 EU countries. To assess the overall security of a website, we measure the adoption rate of 8 countermeasures as indicators of "security consciousness", as well as the presence of 10 common vulnerabilities and weaknesses.

Out of the 22,851 analyzed websites in September 2013, we found that 46.12% enabled at least one security feature. At the same time, we found that 56.39% of the websites contained at least one vulnerability or weakness. Moreover, we found several instances where the website operator tried to protect his website (e.g. against ClickJacking or SSL stripping attacks), but failed to do so by using an incorrect directive or not using the security header over a secure channel.

Due to legal and ethical considerations, our analysis of vulnerabilities in websites was limited to a passive analysis, with a few exceptions. Consequently, the results of the analysis provide only an estimation on the state of security of European websites. However, a preliminary cross-check with sources of vulnerable websites (suffering from Cross-Site Scripting vulnerabilities), and a set of deemed secure websites (i.e. 20 respectable banking websites) indicates that the metric used in the study is able to differentiate between vulnerable and secure websites.

Dataset 2: September 2015

In Section 8.2, we reassessed the same websites in September 2015, and investigated how the overall web security did evolve over time, and which websites did improve, either by applying the security features more consistently over their domain or by applying new security techniques as part of their defense mechanism.

Based on analysis of available data of the two crawling experiment, we could observe the following longitudinal trends:

First, we have observed that the most popular websites (according to the Alexa ranking) have a higher web security metric that less popular websites. Moreover, we could validate that the most popular websites were adopting new security features quicker that less popular websites in the two year timeframe.

Second, the United Kingdom, the Netherlands and Germany are the three best performing countries in EU with respect to the use of web security features, both in 2013 as well as in 2015. Moreover, the Czech Republic and Poland entered the top 10 in 2015, at the cost of Spain and Portugal. In addition, we identified a trend that websites in countries scoring better in 2013, also tend to adopt more new security features in 2015.

Third, by examining the websites based on their business vertical, we can state that the websites in the Finance and Education category are outperforming other verticals in the data set, with respect to the web security metric. In addition, we identified a trend that websites in business verticals scoring better in 2013, also tend to adopt more new security features in 2015.

4.2.2 Third-party Security Seals

As discussed in Section 8.4, providers of security seals claim that websites that make use of their services will appear more trustworthy to the eyes of consumers and will thus have an increase in their sales. In our empirical study, we put the security guarantees of seal providers, i.e., the guarantees that indirectly influence a consumer's feelings of trust, to the test. Through a series of automatic and manual experiments, we discovered that third-party security seals are severely lacking in their thoroughness and coverage of vulnerabilities. We uncovered multiple rudimentary vulnerabilities in websites that were certified to be secure and showed that websites

that use third-party security seals do not follow security best practices any better than websites that do not use seals. In addition, we proposed a novel attack where seals can be used as vulnerability oracles and describe how an attacker can abuse seal providers to discover the exact exploit for any given vulnerable seal-using website. Overall, our findings show that current state-of-practice of third-party security seals is far from ideal. While we propose steps that seal providers can take that will substantially increase the accuracy and effectiveness of their security certification, the issue of inadvertently creating a vulnerability oracle seems to be central to the current architecture of security seals and appears to not have a technical solution which does not sacrifice, either the honesty of a seal provider, or the security of the certified website.

4.2.3 Client-Side Complexity

Finally, in Section 8.3, we presented the results of a comprehensive and through practical study on the security implications of growing client-side code complexity, using the vulnerability class DOM-based XSS as the object of the study. Based on our results, we can conclude the following observations:

- **Unhandled client-side complexity:** The security implication of complex client-side JavaScript code are not properly handled by the developers of Web application, as we uncovered DOMXSS vulnerabilities in 10% of all surveyed sites. Given the comparatively well understand nature of XSS (opposed to novel threats such as Click/LikeJacking or PostMessage spoofing), this does not bode well for less well explored vulnerability classes.
- **Universal problem:** The observed problems apparently occur regardless of page popularity or origin.
- **Tip of the ice berg:** As mentioned above, by any means, the reported numbers are merely the lower bound of real client-side vulnerabilities. The real number (especially taking non-XSS issues into account) is most likely significantly higher.

Thus, in the future to come, all stakeholders in web security (practitioners, researchers and standardisation) have to revisit the presented problem and work on mitigations to handle the growing class of client-side vulnerabilities.

4.3 Observable Gaps between the State-of-the-Art and the State-of-the-Practice (DS.3)

The overview of the Web security threat landscape clearly illustrates the complexity of the Web ecosystem. To improve the end-to-end security, it is necessary to raise the bar on several (if not all) topics in parallel. For instance, in order to fully protect *application transactions*, at least 17 attacks need to be mitigated.

In addition, there exists a remarkable mismatch between state-of-the-art mitigation techniques and best practices being available for almost all vulnerabilities, and we measured only a limited adoption of the best practices in the state-of-practice. The question remains as to how web site owners can be incentivized to actually deploy best practices on their sites? Similarly, to track the adoption rate over time, it is important to have good metrics and measurements in place to be able to assess the state-of-practice of the Web ecosystem.

As part of the analysis, four major trends have been identified that will impact web application security research for the coming years:

#1: Trend towards server-driver browser enforcement. The server defines the security policy, and the browser has the enforcement mechanism in place to enforce the policy.

#2: Clear shift from purely technical to user-centered. Future web security research needs to embrace an interdisciplinary approach as attackers focus more and more on the end-user, and mitigation techniques become hard to understand for end-users.

#3: Focus on pervasive monitoring. Since Snowden, the security community puts much more emphasis on pervasive monitoring and man-in-the-middle attack scenarios, both for exiting as well as new technologies

#4: An increasing need to compartmentalize web applications. As web applications are becoming larger and contain more third-party components, the secure containment or sandboxing of untrusted parts of the web application becomes a crucial factor in secure the web applications.

4.4 Interactive Survey (DS.4)

The interactive survey (see Chapter 10 confirmed to a large extend the assumptions STREWS made based on other data sources and via the constant communication within IETF, W3C, research and the Commission initiatives around Cybersecurity. A confirmation is that STREWS did well in taking up emerging Web technologies for its use cases.

A central question in the survey, which was mainly answered by professionals with background both in software engineering and security, was on perceived upcoming security challenges, with the following results:

- *Online tracking:* Tracking and content injection by the advertisement industry was seen as one of the biggest threats to Web security. This in turn facilitates the creation of a technology that creates ideal conditions for pervasive monitoring.
- *Re-centralisation:* The above goes hand in hand with the re-centralisation of the Internet and the Web by large corporations controlling large parts of the technology stack was identified as a threat to security and privacy
- *Complexity:* The complexity of the system was blamed in many responses. This included the fact that deployment of solutions on the Web is very costly, burdensome and long.
- *Development model:* Lack of testing and an environment that favours design over security and resilience make it harder to find the necessary momentum.
- *Security UI:* The lack of good user interfaces for security is a concern.

Thus, the main new aspect that resulted from the survey's answers is certainly the extend to which tracking is seen as an issue and even interfering with security. Furthermore, while the focus of this roadmap is on research, it has to be noted that the survey exposed an astonishing clarity of the security experts with respect to the re-centralisation of the Web by large corporations. This raises a political issue that can not be underestimated. Similar to the monopoly in economics, there is a risk of a very high influence by certain stakeholders that is abused to optimise for their purposes the technology that even countries depend upon. This conflict of interest needs certainly more research to allow for an informed decision making in the future. It is essential for the deeper understanding of a Cybersecurity policy that can have success.

4.5 NoE and Policy Activities (DS.5)

The analysis of the work of Networks of Excellence in the area of security showed a broad similarity of the topics that can be found in Roadmaps and enumerations. This results in a list of trendy security topics like quantitative assessments, security usability, software development

life cycle management and security by design. This list has no tangible relation to the challenges addressed by the Cybersecurity strategy discussion. Cybersecurity, on a political level, is seen rather as a challenge to emerge a security response administration and to create the necessary conduits so that EU member states talk to each other in case of a security incident. While the political documents talk about securing our critical infrastructure, it remains unclear what is covered by «*critical infrastructure*». Those concerns, in turn, are not related in a comprehensible and clear way to the topics trending in the security Roadmaps. This is true for research but also for NIS. Both of them have a very distant relation to Web Security or even Internet Security. While every introduction evokes the ubiquitous presence of networks in our lives, no document actually makes the link to the fact that those ubiquitous networks are mainly the Internet and the Web. What remains is a feeling of isolated and sometimes repeated thought-silos that are themselves, unrelated to latest developments in Web Security and Internet Security. But the situation is not hopeless. Because some of the new ideas from research would merit to find their way into large scale application on the Web. The issue though has a technical and a social dimension. While a certain security technology may exist in isolation, it is often not integrated into the platforms used by everyone. And the challenge remains, that researchers gain merit from the writing of original papers, not from fixing actual problems of the Internet. Thus no wonder that the security community is segregated that way.

4.6 STREWS Workshops (DS.6)

The STRINT Workshop was a very successful event in terms of outreach and awareness. A hundred people attended and the room could have been filled twice. The who-is-who of Internet development was in the room. A representative of the EDPS attended. The audience was very high level. But the problem of pervasive monitoring is a hard one. So nobody expected quick and easy solutions. STRINT was a first step to start a deeper discussion that goes beyond an IETF plenary debate that must remain constraint in time and scope. People presented ideas. There were good ideas and not so good ideas. As a main result, people in the IETF now think harder about data minimisation, which is a huge step forward with respect to the European data protection approach. The second central point is that communications will be encrypted. New protocols or protocol revisions will have to justify their aim to remain in the clear. A major issue were middleboxes. The most problematic use case are hotel networks that derail DNS and other traffic and inject all kinds of junk. The appeal to those networks to change their technology and not alter content in transit was made. Finally, all week, the IETF meeting and their over 700 participants heard a lot about STREWS and STRINT. Workshop and project were mentioned in the conclusions of Jaari Arkko, the chair of the Internet Engineering Steering Group.

4.7 Standardisation (DS.7)

The activities on Cybersecurity and Information Security in all Standards Bodies we looked at were manifold. There is a rather clear repartition of work between most of the Standards Organisations around the Internet. There is some overlap here and there and there are competing activities, even in Security. The IETF has traditionally the biggest slice of security work in the internet stack. W3C continues to explore avenues to add security tools to the Open Web Platform. IEEE is very much focused on the hardware layer, including low level networking protocols. With SAML and XACML, OASIS has accomplished some important work. The integration into the Internet stack remains unclear as the Web moves to an application platform that is less based on XML and more oriented toward JSON serialisations.

4.8 WebRTC case study (DS.8)

The case study and the related STREWS activities (e.g., the special issue of the IEEE Computing journal) uncovered, amongst others the following emerging security challenges:

For one, an important aspect in securing the client-side WebRTC environment, is the ability to enforce security policies in the JavaScript execution context. As for now, the JavaScript code that controls the whole WebRTC communication runs in the browser of the end-user (and executes on his behave), but is at the same time under control of the calling page, and it will most probably include (and delegate control to) third-party libraries. Put together, this opens an an interesting attack vector for various attacks (including injection attacks) while receiving and interpreting data via the signaling or media path, and running code from the calling site and third party script JavaScript providers.

Another aspect is the overall complexity of the technology: the inherent complexity in setting up the peer-to-peer connections between browsers, and the combination of two rather distinct worlds – each with their own security model: end-to-end networking and JavaScript/Web. In particular, we expect the origin-based security model of the Web to conflict with the connection-based security model of real-time communication. For instance, questions arise as how to setup a connection across different JavaScript origins, or how to manage multiple connections and identities from within a single origin and JavaScript execution context.

Moreover, several technical solutions on the identity provisioning side as well as on the JavaScript permission side propose to enroll the end-user is making security-sensitive decisions (e.g. whether to allow the browser to set up a call with user@idp (once or permanent), whether to share his video stream or desktop, or whether to trust site A in calling external parties, ...). This will undoubtedly expose (some of) the complexity of the technology to the end-user, but even more critical this also implies that the end-to-end security of WebRTC applications will strongly depend on the judgment call of a novice end-user, potentially "promoting" this end-user to the weakest link of WebRTC.

In conclusion, while being founded on a significant body of security considerations, WebRTC will keep security-minded professionals occupied for the foreseeable future nonetheless. Given the many facets and complexity of the underlying communication and application model, this is hardly surprising.

4.9 Web Security Architectures Case Study (DS.9)

The second STREWS case study yielded a considerable amount of insight in the current state of web security architecture, potential future directions and unsolved problems.

4.9.1 Recent Developments

For one, the case study identified several noteworthy upcoming technologies within the web stack, that could potentially play a role in addressing emerging security threats of the modern web. In particular, these technologies span the following areas:

- **The Networking and HTTP layer:** From the area of standardization, mainly driven by the IETF, various incremental and fundamental improvements to the network-portion of web applications are emerging. In the case study, the following developments were identified to be particularly noteworthy:
  - *Privacy Enhanced IP Addressing* and *Transport Layer Security* improvements on the network layer,
  - *DNSSEC* and *DNS Privacy* on the DNS layer,
  - and *HTTP version 2* as well as *HTTP Origin Based Authentication (HOBA)* on the HTTP layer.

- **Server-driven security policies:** A set of emerging technologies tie the server closer to the client, via security policies that are pushed by the server directly to the browser. These mechanisms include
  - on the protocol layer: *X-Frame-Options*, *HSTS* and *Key pinning*,
  - and on the application layer: *Content Security Policy (CSP)*, *User Interface Security Directives for Content Security Policy* and *Entry Point Regulation for Web Applications*.
- **New security-centric JavaScript APIs:** Finally, a multitude of new JavaScript API that either directly address security concerns or are security sensitive are currently under active development. In this field, we can identify two major clusters:
  - Secure collaboration: APIs for *Protected Document Exchange*, *Cloud Storage*, *Document Signing*, *Data Integrity Protection* and *Secure Messaging* have the potential to enable sophisticated multi-party application scenarios.
  - Authentication and encryption: The upcoming APIs for *Credential Management*, *Multi-factor Authentication* and *Web Cryptography* will bring fundamental improvement in these area to the Web browser.

Challenges for upcoming technologies: A major challenge for the majority of these emerging standards, is reaching significant uptake by operators of web application infrastructures, namely sever-operators and deployers of the actual web applications. Furthermore, for a subset of the proposed technologies, existing infrastructure actively hinder their adaption. E.g. despite the importance and usefulness, DNSSEC still lacks ubiquitous deployment. This is not only due to the difficulties in implementation, but also due to the fact that there are a range of middle boxes redirecting people that would be put out of business with DNSSEC. The governmental blockings are only one examples for certain techniques that will not work any more once DNSSEC is deployed. More importantly, most Hotel networks today depend also on the manipulation of the DNS request to redirect people to the payment portal or to some click-through page that contains disclaimers and terms of service.

In the field of server-driven policies, the user interface to security remains by far the most difficult issue to tackle. The W3C Workshop on Privacy and User Centric Controls has shown that issues related to implementing and achieving adoption related to privacy and security may be similar to those for accessibility and internationalization. The latter are seen as a constant challenge and review- point for all specifications. The Workshop has raised the right questions, but wasn't able to come up with agreed answers despite the number of high quality contributions. It has been noted earlier already that the user interface is where browsers compete. So agreement in this area is very difficult as it faces to constant challenge not to overstep the line where space is reserved for competition between the actors.

However, as it is with all newly introduced APIs, only thorough, independent security assessment, as done with the STREWS WebRTC case study, can verify that the APIs' security promises indeed hold. We foresee necessary major effort on this in the future.

However, the Workshop on Web Cryptography Next Steps concluded that further work is needed to address authentication and the integration of hardware tokens. But the current Web Cryptography Working Group charter does not allow for the integration of hardware tokens. In the subsequent chartering discussion, the Working Group decided against developing an API to access security hardware elements. This is bad news for Europe, as access from the browser and the Open Web Platform to the security features of Identity cards will be more difficult and will require dedicated software.

4.9.2 JavaScript Sandboxing

Within the second case study, JavaScript sandboxing was identified to be an essential building block to ensure the security of next generation, multi-party web applications. The case study that there is a large body of research into JavaScript sandboxing solutions, but that there is not a clear winner in terms of the perfect solution.

- *Sandboxing with browser modifications:* Browser modifications are powerful and can sandbox JavaScript efficiently, because of their prime access to the JavaScript execution environment. Unfortunately, the software modifications are difficult to distribute and maintain in the long run unless they are adopted by mainstream browser vendors.
- Sandboxing without browser modifications: JavaScript sandboxing mechanisms without browser modifications leverage existing browser functionality to isolate and restrict JavaScript. This approach can be slower and less flexible but requires no redistribution and maintenance of browser code. In addition, it automatically works on all modern browsers.

From the review of relevant research, we can distinguish a number of components that are always present in a JavaScript sandboxing solution, namely support of Virtual DOM creation, an *isolation unit*, and an expressive policy language. While the policy language can be realised with current means, the other components require future attention to mature sufficiently:

- **Isolation unit:** None of the existing technologies, namely Browser Extensions and Web Workers are sufficient to implement a functional and secure isolation unit within a web application. Thus, further research and development in the realm of web browsers is required to unite the two techniques and outfit them with a matching proxy mechanism, that connects the sandboxed code to the main web application.
- **Support for Virtual DOM creation:** The virtual DOM needs to reflect the methods and properties available in the real DOM as accurately as possible, because running JavaScript code has certain expectations with regard to the appearance of a "standard DOM". Using the Proxy API to wrap the real DOM, presents executing JavaScript code with an apparently real DOM while at the same time allowing full mediation. However, currently using the Proxy API, the construction of a virtual DOM thus involves a full description of the real DOM, which is not available through any JavaScript API. Such a description has to be imported from elsewhere, as JavaScript code, JSON or even IDL listings. Thus, since the browser itself has the information about the structure of its own DOM, it should not be necessary to import it from elsewhere. Sharing this information from the browser in a structured way through a JavaScript API, could ease the construction of a virtual DOM.

  This essential support in currently missing in the available set of browser capabilities.

In conclusion: The current generation of browsers is not capable to enable secure JavaScript Sandboxing. While selected individual scenarios can occasionally be realised using the existing browser capabilities, universal and powerful sandbox requires further developments of the Web's client-side.

4.9.3 Cross-site Scripting

Finally, the second STREWS case study showed, that Cross-site Scripting (XSS) is as much a problem today as it has been ten years ago, when the first wave of attention to XSS was paid by the security research community. XSS is as much evolving as the web platform on which the vulnerability class manifests itself.

Most essential, we identified several open problems that are not well-understood so far and where additional research is needed:

- **Client-side XSS** As it was demonstrated in 2013 by Lekies et al [125], client-side XSS is a widespread problem in modern Web sites. In this study, the authors found DOM-based XSS vulnerabilities in roughly 10% of the Alexa Top 5000. However, only a small number of the approaches explicitly consider this specific subclass of vulnerabilities and it remains to be evaluated, to which degree existing approaches can be adapted to the specifics of client-side XSS.
- **XSS Outside of the Browser** Ever since HTML was adopted to create UIs of general purpose applications, XSS is no longer confined to the Web browser, affecting native application, operating system dashboards, mobile applications, or IoT appliances. Up to now, only little systematic research has been conducted on XSS outside of the browser. However, given the growing importance of this development approach and the potentially enhanced privileges of injected scripts, the topic warrants further attention and should be studied in detail.
- **Content Security Policy (CSP) and Script-less Attacks** CSP has the opportunity to become a major factor in the battle against XSS exploits, in case of substantial adoption of the standard in the future. However, CSP only mitigates JavaScript injection, while markup injection still remains possible. For this reason, offensive research emerged, that focus on XSS exploits that do not rely on JavaScript execution. For instance, Heiderich et al. [96] showed how to leverage HTML/CSS injection in combination with SVG fonts to leak sensitive data from an application. Additional information exfiltration attacks that function *without* scripting have been documented by Chen et al. [49] and Zalewski [222]. The capabilities and likelihood of such attacks are still subject to future research. Furthermore, defenses against script-less attacks are needed.
- **Self-XSS** The term *Self-XSS* describes a class of social engineering attacks in which a user is tricked to copy & paste `javascript:`-URLs into the browser's address bar, causing the browser to execute the contained script in the context of the top-level document. As JavaScript's syntax allows several obfuscation tricks (e.g., encoding the full script without alpha-numeric characters [95]), the nature of the to-be-pasted URL is easily hidden and social engineering attacks are viable. On a technical level, Self-XSS offers the same offensive capabilities as all JavaScript injection techniques. Hence, it has to be rated on an equivalent severity level. However, up to this point only little research has been done in respect to applicable defensive measures and thus research on containing and mitigating such injections is necessary.

In conclusion, XSS is far from being a "solved" problem. Given the severity with which a single successful XSS attack thoroughly compromises the client-side of a vulnerable web application, it is paramount to investigate into a web platform that is resilient against such threats.

4.10 Cyber Security (DS.10)

The Cybersecurity has certainly its merits and helps to organise the relevant structures and communication channels within Europe. But so far, Web Security or Internet Security are not well served. But Web technologies are crucial for the implementation of all those measures. It is therefore of utmost importance for the success of all those Cybersecurity measures to be harmonized and coordinated with the development of the Web and the Internet at large. Research and implementation play a decisive role. In WebRTC, STREWS has made a significant contribution to the overall security of the Web platform. But it is not sufficient. If the Commission wants the eIDAS scheme to succeed, it has to invest significantly into its integration into the Web and the Internet. This means participating in the work around TLS v.1.3, but also a high attention to the proposed work on Hardware Security in W3C[210]. If the Web and the Internet are not

treated as a first class citizen in the eIDAS and Cybersecurity initiatives, we will see a parallel world where the measures play a certain role in enterprise computing, but are not visible or addressable by the public at large who uses the Internet and the Web. The enrollment at the University Torino will either require an expensive special device or it will be still done in paper. With investment into the Web integration, the eCommerce in the Digital single market, in turn, will receive a boost in confidence that makes European wide transactions simple and secure and will contribute greatly to growth. This includes work on connecting Web applications and trust services on a technical level and to integrate the identity scheme created by eIDAS into the Web and other Internet services, e.g. using jabber.

Chapter 5

Overarching Conclusions

In this chapter, we systematically revisit the roadmaps five objectives (see Section 2.2) to draw overarching conclusion from the collected data, to guide the future of web security research in Europe.

5.1 Significant gaps between the state-of-the-practice and current research results

In this section, we examine (OBJ.1) *"Identify significant gaps between the state-of-the-practice and current research results"*. For this purpose, the main datasources were: (DS.1) *"The State-of-the-Practice in today's Web Software"* (see Chapter 7), (DS.2) *"Selected Empirical Studies"* (see Chapter 8), (DS.3) *"Observable Gaps between the State-of-the-Art and the State-of-the-Practice"* (see Chapter 9), and (DS.4) *"Interactive Survey"* (see Chapter 10), with the remaining datasources providing additional insight when applicable.

As it was reported in the external vulnerability statistics (DS.1), today's web applications are still heavily susceptible to classic vulnerabilities, such as SQL Injection and Cross-site Scripting. The numbers of vulnerable websites, a number between 77% and 96%, is noteworthy, with SQL injection being on a high ranked position.

The empirical study on client-side Cross-site Scripting (DS.2) resulted in the same insight: Approximately 10% of all examined sites are susceptible to this narrow security problem that only affect a small fraction of an application's code base. Client-site XSS is only one variant of a single vulnerability class. Thus, the likelihood of similar numbers for related vulnerabilities is high. As we reported in the context of (DS.3): For instance, in order to fully protect application transactions, at least 17 attacks need to be mitigated.

Many classic vulnerability classes, including SQL Injection, are considered by the research community to be "solved". For instance, the common assumption in respect to SQL Injection is that through using "prepared statement" and persistence frameworks, web applications are immune against SQL Injections. If this is the case, why do we see so many SQL Injection vulnerabilities in real world applications? Are the proposed solutions not as universal as assumed or is their protection surface incomplete? Or are other factors hindering the protective technique's adoption?

5.2 Alignment of practitioners' needs and research/standardisation

The second motivating objective (OBJ.2) of the roadmap's data collection process was to *identify mismatches in between ongoing/future standardisation and research activities in the field of Web*

security and the needs of the Web's practitioners. Here the major data sources were: (DS.1) *"The State-of-the-Practice in today's Web Software"* (see Chapter 7), (DS.2) *"Selected Empirical Studies"* (see Chapter 8), (DS.3) *"Observable Gaps between the State-of-the-Art and the State-of-the-Practice"* (see Chapter 9), (DS.4) *"Interactive Survey"* (see Chapter 10), (DS.5) *"Review of related NoE and Policy activities"* (see Chapter 11), (DS.6) the *"STREWS Workshops"* (see Chapter 12), and (DS.9) the second STREWS Case Study on *"Web Security Architectures"* (see Chapter 14) with the remaining datasources providing additional insight when applicable.

As it was discussed in the previous section, a significant fraction of all web sites suffer from "classic" problems, such as SQL Injection or server-based Cross-site Scripting. Researchers, under the assumption that these problems are understood and solved, moved on to other web security related topics. E.g., as it is documented in the second STREWS case study, the research work on preventing classic XSS has mostly stopped several years ago and XSS is now mostly examined in respect to new attack classes or for smaller subsets, such as purely client-side XSS.

However, the security statistics clearly show, that the classic problems are not solved as many new "classic" vulnerabilities enter the landscape on a daily basis. Thus, web security research should aim to examine why the proposed solutions are not sufficiently utilized and, potentially, develop better solutions for the classic vulnerability classes.

Instead the current focus of web security research (and consequently also of standardisation) is on web security problems that are perceived to be more novel or more advanced, such as ClickJacking, JavaScript messaging or DOM-based Cross-site Scripting. As as result, for several of these advanced web centric security problems, potential remedies were introduced in the last years, including server-driven defense against Cookie Theft, ClickJacking or SSL stripping attacks. However, as we documented in Chapter 8 (DS.2), the uptake of these protection techniques is lacking in the European web. For instance, only 33.51 % of all examined web sites utilize `HTTPOnly` cookies, a easy to adapt session security mechanism that exists since 2007. Only 4.5 % defend against ClickJacking using the `X-frame-options` header and only a mere 0.5 % of all sites aim to prevent SSL Stripping Attacks via the `Strict-Transport-Security` HTTP response header. Even though a light increase of usage in the European web could be observed between 2013 and 2015, the overall adoption numbers are still very low.

None of these mechanisms requires significant deployment or implementation work (unlike for instance an introduction of Content Security Policies (see DS.9), which entails application changes). Hence, similar to the "classic problems" the question arises: Why don't we see higher adoption numbers?

5.3 Emerging topics and future hot spots of web security

The roadmap's third objective (OBJ.3) is to *"identify the emerging topics and future hot spots of Web security"*. Using this document's data sources, with special attention to the selected empirical studies (DS.2) (see Chapter 8), the interactive survey (DS.4) (see Chapter 10), the STREWS workshops (DS.6) (see Chapter 12) and the STREWS case studies (DS.8 & 9) (see Chapter 14), the following emerging topics in the area of web security could be identified.

5.3.1 Client-side Complexity

A through-line over all empirical data sources (DS.1, DS.2, DS.9) was the observation, that the web platform currently undergoes a fundamental shift towards moving application functionality to the client-side, i.e., to the web browser's JavaScript. This is the cause for a significant number of security vulnerabilities (DS.2) and renders existing server-centric security approaches useless (DS.9). Furthermore, this shift will only increase with the ever growing set of client-side capabilities in the web browser. As shown in the first STREWS case study (DS.8), the web browser is now able to host full-featured audio and video chat applications, that either directly connect browsers peer-to-peer or interface with non-web technologies, such as SIP or XMPP. The

standardisation bodies (DS.7) plays an active role in this development through a full pipeline of novel powerful JavaScript APIs (DS.9).

5.3.2 JavaScript Sandboxing

As explored in the STREWS case study 2 (DS.9), JavaScript sandboxing is an emerging topic, which has net yet been realised to its fullest potential. Sandboxing potential untrusted content in isolated web container can improve the overall security of a web application. Isolating third party scripts, which can be frequently found in web pages, as seen in one of (DS.2)'s empirical study, can reduce potential unwanted third party user tracking, a demand that was recorded in the interactive survey (DS.4). And last but not least, JavaScript sandboxing can be a powerful tool to mitigate client-side vulnerabilities, such as DOM-based XSS (DS.2), through isolating potential vulnerable code from the trusted computing base of the application.

5.3.3 Server-driven Security Policies

Server-driven security policies is a promising tool to overcome the web platform's client-server gap: Using such policies, the server can instrument, enhance and steer the browser's process how to interpret the received HTTP responses. The concept of server-driven security policies is reoccurring in the context of various topics the data gathering process covered: For one, various of the recent and upcoming improvements to the web platform's network layer (see DS.9), which are currently under active standardisation (see DS.7), utilize the concept, such as HTST, Key Pinning or X-Frames-Options. Furthermore, the server-driven Content Security Policies are among the most promising measures to mitigate XSS vulnerabilities (DS.2 and DS.9). And finally, the emerging JavaScript sandboxing approach (DS.9, see above) utilizes server-driven policies, so that the browser has guidance how to handle the untrusted script content.

5.3.4 JavaScript Crypto and Hardware Tokens

With the trend towards moving the application logic onto the client (DS.3, DS.9), the requirements for realising security features grow. An especially important topic in this area JavaScript Cryptography. As soon, as sophisticated authentication or interaction protocols will be implemented purely in the browser space, cryptographic routines are necessary. For this reason, a native cryptographic API (DS.9) is currently under active standardisation (DS.7), so that applications do not further have to rely on incomplete external JavaScript libraries, such as the Stanford JavaScript Crypto Library [189].

Having the raw algorithms is only the first step. To fully integrate applications that run on the web platform into larger scenarios, such as eGovernment, it is required that hardware authentication tokens, as they for instance can be found in ID cards, can be integrated in the application's workflows. Furthermore, access to trusted execution environments and other hardware based security systems is highly desirable.

5.3.5 The end of the client-server paradigm

As seen in the first STREWS case study (DS.8), with the introduction of WebRTC browser can now interact directly in a peer-to-peer fashion without intermediate web servers. While the initial motivation for WebRTC probably was mainly to enable A/V chat solutions in the browser, it actually brought a significant break from the established strict client-server paradigm of the Web. Using WebRTC's data channel capabilities, it will be possible in the future realise application scenarios, which were impossible only a couple of years ago.

In a similar fashion, the usage of web technologies in non-browser environments, such as operating system dashboards or mobile applications, does not only bring classic web vulnerabilities (as seen in DS.9) to these execution environment, it also is a departure of the client/server

model. Furthermore, it challenges many of the web platform's most fundamental security pillars: What is the origin of JavaScript code in a mobile application and how should the same origin policy be applied? And to which degree can the authenticity guarantees of TLS be maintained, when two end-points communicate without intermediate servers?

5.3.6 Web privacy

It can be concluded from the interactive survey (DS.4), a major security concern for web users is privacy and user tracking. The European research on Web privacy has been very successful in the past. But research in this area is disconnected from the standardisation and development worlds.

With ABC4Trust we know more about anonymous credentials, but we do not know how those will be integrated into the technical platform that we use in our every day life. This means we discovered quite a disconnect between the privacy research and the current activities in standardisation. While research is either focused on legal issues in privacy or highly complex cryptographic and technical solutions, the standardisation fights with the problems of every day life. Consequently, standardisation is not particularly helped because the solutions provided by research are not usable for the issues at hand. And the people in research take their merit from presenting cutting edge solutions with proof of concept island implementation. Participating in standardisation will not increase their merits. And the standardisation people have no interest to look into research as it is disconnected from their world of real browsers and business models.

Furthermore, new challenges like privacy on the Internet of Things are currently not researched at all. This will lead to a situation where those creating the Internet of Things will put privacy into their Specifications to the best of their knowledge. And this will determine the landscape that later requires patching for privacy instead of privacy by design. Once again, WebRTC may serve as an example. While privacy by design would require a good transparency of the permissions given to scripts to use camera and microphone, browsers will currently implement a system that only asks once for permission and stores that forever. A research that creates evidence into the standardisation move that this leads to further damages and goes against the Cybersecurity and resilience - strategies is not in view.

One of the more interesting new developments is that people start to think about standardisation around what requirements and features the browsers private browsing mode should fulfil to call themselves private browsing.

5.3.7 Advancing web authentication and session tracking

As thoroughly explored in the second STREWS case study (DS.9), secure web authentication and session management is paramount for the security of a given web application. Due to the involvement of both server and client in this area, fundamental advances are only feasible if practitioners, browser vendors and standardisation make a coordinated effort to adapt existing and future research results. The current standardisation work on HTTP/2, DNSSEC and DANE (DS.7) can already be counted as steps in this direction.

5.4 The state of standardisation in relationship to the web platform's emerging security topics

In this section, we explore the roadmap objectives (OBJ.4) *"Map standardization and research efforts to the observed emerging topics in Web security. Identify topics that require further attention or are not covered yet"*. For this purpose, Section 5.3's list of identified emerging topics was cross checked with the standardisation related results of (DS.7) and (DS.9).

Research and standardisation are mostly aligned

Many of the topics in research find a corresponding effort in standardisation in the Web Application Security Working Group. It is well known that the mismatch between the state of the art on the one hand and research on the other hand is often due to the installed base that isn't updated. SQL injection is the best example of this top concern. There is no research on how to overcome this issue. But also standardisation is lagging behind slightly. There is a good exchange of views. After the rechartering of the Group, it also works on confinement. Most of the work is inspired by the OWASP Top 10 List mentioned in this document.

Well aligned areas:

In particular, good alignment between web security research and standardisation can be observed for the following emerging topics:

- *Server-driven policies:*

  To address client side complexity the standards that create this complexity already address some of the research results in their requirements and security considerations. There is a constant dialogue on how to mitigate attacks using the complexity of the setup.

  The Server driven policies are currently at the core of the activities in the Web Application Working Group. There are a variety of Specifications under development that are described in 13.3.2. Further standardisation will need further research into the vulnerabilities of the web platform and its then current tooling including new developments like service worker.

- *Web privacy:*

  As it became apparent in the interactive study (DS.4), a major concern of the web's users lies within the field of web tracking and online privacy. This confirms an observation that already has been made within the STREWS STRINT workshop (DS.6), here in connection with pervasive monitoring. Privacy protection is a well established topic in the research community. Thanks to standardisation activities on the network layer, such as HTTP/2, DNSSEC or DANE, network-based privacy violations are addressed appropriately. In the same fashion, the application layer Strict-Transport-Security HTTP header, which originally was proposed by researchers, will add much needed robustness to the web platform's network connections.

Areas with identified mismatches:

The following topics could be identified, that require further attention or are not covered yet:

- *JavaScript cryptography and hardware tokens:* As discussed above, an extension of JavaScript cryptography toward hardware tokens, trusted execution environments and other hardware based security systems is required to realize non-trivial application scenarios, such as eGovernment, securely. However, as discussed at the Workshop on Web Cryptography Next Steps (DS.6), none of these capabilities fit the current charter of the working group. In the subsequent chartering discussion, the Working Group decided against developing an API to access security hardware elements. This is bad news for Europe, as access from the browser and the Open Web Platform to the security features of Identity cards will be more difficult and will require dedicated software.

- *JavaScript sandboxing:* Researcher advocate JavaScript sandboxing as a powerful tool to mitigate potential vulnerabilities and realized sophisticated multi-party application scenarios, in which the browser is utilized to host the connecting middleware. However, as documented in the second STREWS case study (DS.9), current browsers are lacking support to natively realise powerful sandboxing solutions. And unfortunately, in this area no

promising activities from standardisation exist, that might have the leverage to push the browser vendors into including advanced JavaScript capabilities, such as tools for isolation units or virtual DOMs, which would enable the implementation of flexible sandboxes.

5.5 The state of European research in the field of Web security

Finally, the fifths objective (OBJ.5) was on *obtaining current information on the state of European research in the field of Web security.* For this, mainly the research overviews provided in (DS.2) *"Selected Empirical Studies"* (see Chapter 8) and (DS.9) the second STREWS Case Study on *"Web Security Architectures"* (see Chapter 14) along with the general overviews in (DS.5) *Review of related NoE and Policy activities* and (DS.10) *Cyber Security* were utilized.

In general, compared to major security research fields, such as Cryptograhy, web security research appears to be handled only on a small scale in the European academic landscape. Single university groups, notably for instance at the KU Leuven or Chalmers University, contribute significant scientific works to the area. However, American web security research groups at institutions such as Stanford University, UC Berkeley, CMU or UB Santa Barbara dominate the field due to quantity and quality of their output.

Furthermore, on an coordinated European level, there are not many projects doing research into Web security. SWEPT is one. While SWEPT is a project that addresses Web Security directly, it has a very specific approach that consists of providing a tool for Cybersecurity. This consists of checkers and a platform to combine several checking services. But the tooling is less of a concern for STREWS as it is on the top of the Web platform. But before talking about tools to be connected to the services making the Web platform, we have to research more the vulnerability of the platform itself. This was so far only addressed by STREWS, mainly by the case studies. Those case studies have proven to be hugely impactful. A roadmap would recommend more of that.

Web Security is of course part of the wider security landscape. This wider landscape was explored by several projects, e.g. NESSOS. Those projects issued conclusions and STREWS explored those and tested them against relevance for the Web platform. Only a rather small part of their research was related to Web technologies. Some issues were similar, especially the focus on the user interface issues around security. This constitutes a large field of issues where further research is needed.

While the many initiatives around Cybersecurity will raise awareness, they are focused on creating the necessary administrative infrastructure to secure European systems. But this isn't addressing Web security directly despite the high impact of Web technologies on critical systems within the Union. Additionally the human factor is an important part of security. Better interfaces will thus augment the security of systems considerably. The current research in the area of Cybersecurity concentrates on identity systems that can be used for identification and authentication. But those systems are seen in the context of governmental control systems and do not connect well to the Web. This in turn leads to rather adventurous connections between those systems and the Web. As they are not well researched, they have a higher potential for vulnerabilities. A better integration of Web technologies into the Cybersecurity strategies is therefore recommended.

Chapter 6

Upcoming Security Research Challenges for the European Web

In this chapter, we use the collected insight to pinpoint the upcoming security research challenges for the European web, which either directly result from the emerging topics (see Section 5.3) or materialized as notable insights from the objective-level data correlation of Chapter 5:

- Challenge 1: *Revisiting classic attacks*
- Challenge 2: *Handling the extending web paradigm*
- Challenge 3: *Realizing real end-to-end security*
- Challenge 4: *Increasing End-user Security and Privacy*

The combination of the identified emerging topics and the listed overarching research challenges results in an exiting and promising research roadmap for the mid to long term. We expect, that following this roadmap will lead to potentially impactful results, which address the future security problems of the Web, while being well suited to be adopted by practitioners and standardisation.

To achieve these ambitious goals and foster the impact of the results, another STREWS project is needed to bridge the gap between Internet standardisation and the various Cybersecurity initiatives. Such a project will have to do an in-depth case study as it was done by STREWS in WebRTC and identify the areas were the Web world and the Cybersecurity thinking are far apart from each other. The project should then suggest concrete actions to bridge those gaps.

6.1 Revisiting Classic Attacks

The correlation of the real world security problems of the web's practitioners community with the current activities in research and standardisation uncovered a significant gap. While research primarily handles novel advanced classes of security vulnerabilities, actually deployed web applications suffer from the same attacks as they did ten years ago. Furthermore, our empirical studies exposed a worrying slow uptake of available security mechanisms by developers and operators of web applications. Thus, research is required to understand and close this gap.

The existing solutions have to revisited and critically examined, why their uptake in practice is so underwhelming. What are the shortcomings of these techniques, that hinder their adoption?

Especially, we need to investigate the problems of injection attacks further as new attacks keep popping up. In the course of this investigation, it is necessary to come up with novel countermeasures for both client and server, that not only address the vulnerability but also overcome the currently existing adoption hurdles.

As this is a non-trivial goal, that likely requires long-term time investment and potentially radical changes to the web platform, it is also important to target short term results to generate timely effects. For this an a promising approach is to investigate incremental changes to the web security policies, that can be easily adopted by standardization bodies and browser vendors. This approach is in line with more recent, existing results on X-Frame-Options, HSTS, or the Content Security Policy.

Finally, to closely monitor the development in the field and to measure the success of future security research activities, there is a need for more ongoing experimental research to capture the widespreadness of weaknesses, and measure the adoption rate of security features (e.g., over time and website category).

6.2 Handling the Extending Web Paradigm

As reported in this deliverable, the web platform is ever evolving and changing. The recent years have brought a significant shift towards powerful client-side capabilities. Fundamental circumstances that defined the boundaries of the platform and that appeared to be set in stone, get softened or disappear completely:

- For instance, while previously being bound to strict client-server HTTP communication, web browser can now communicate client-to-client via DTLS peer-to-peer connections, thanks to the introduction of WebRTC.
- Furthermore, the Same-origin Policy that confines a JavaScript to the domain of its rendering document is the main pillar that client-side web security resides on. However, more and more new additions to the platform and APIs, such as Web sockets, CORS, PostMessage, or Service Workers, enable JavaScript to widen the confines of the Same-origin Policy or leave it completely.
- Finally, even the assumption, that web code is executed in the context of a *web application* is no longer true. Instead, web technologies are utilized in various non-web contexts, such as mobile applications or dashboards. With FirefoxOS phones and Chromebook laptops even two device classes have been introduced, that use the web platform as the basis for their operating system.

With the negation of more and more of properties that previously were assumed to be static and reliable, also a lot of preexisting security assumption need to be reevaluated.

Furthermore, as browsers and web applications are more and more evolving towards a general paradigm that replaces operating systems and applications, it is important to invest and mature the security fundaments in the web application architecture. This includes investing in research and development on access control, information-flow control and meta-data architectures.

Finally, WebRTC and other advanced web technologies may be pushing the web origin model of security used in modern browsers to it's limit. There is a need for research into the real effects of this model and into practical ways in which this model could be extended or eventually (over decades) superseded. It is crucial that such research pays attention to the reality of deployments and implementations - a totally clean-slate is just not available here and any transition will require changes to both browsers and web content.

6.3 Realizing Real End-To-End Security

In the current web platform, a severe gap between the server and the browser exists. While the server is under tight control of the web application's operator, the browser is in large parts an unknown entity. The server can push web content to the browser and hope the the browser's interpretation of this content matches the server's intend. But there are no guarantees that the

server's assumptions hold true. This is especially problematic in the presence of vulnerability classes, such as XSS or ClickJacking, and made even more complicated in multi-party application scenarios with effectively use the web browser as connecting piece between to mutually distrusting web servers.

To handle this problem, website security needs to be addressed holistically as an end-to-end solution. More specifically, potential approaches should provide client and server side security fundamentals and building blocks, express web security policies end-to-end, and make abstract security primitives directly available to website architects and developers. This implies automatic selection and deployment of security building blocks across the end-to-end web system.

6.4 Increasing End-user Security and Privacy

Up to now, web security is frequently regarded as a client-server security problem. This narrow and overly technical view often ignores the security and privacy needs of the participant in the application model that should be the center of attention: The end-user that operates the web browser.

End-user security: For one, in the future attention should be paid to techniques, that allow for stronger end-user authentication and deprecate the established practice of bearer token authentication. In this context, user interfaces need to be investigated that allow to gather multifactor security context information to be scrutinized and controlled by the user within a browser. Furthermore, in order to allow for a smooth integration of the eIDAS Regulation into Web technologies, more research on browser interfaces (APIs) to hardware are needed. This includes the secure contextualisation of the authentication information issued normally by such security hardware for the Web interactions.

Secondly, targeted and mass attacks that directly affect web users are a concern. Hence, more research is needed in the field of reactive measures, that help users to combat web attacks due to insecurely implemented web applications.

End-user privacy: The community input in the form of the interactive survey and the STREWS workshops has shown: User tracking and pervasive monitoring is a severe concern for many end-users. Thus, more research is needed to make the Web more robust against tracking. This includes pervasive monitoring as well as the data greedy advertisement networks and social media. In this context a ready to go package to use DNSSEC and DANE in the EU would increase encryption between nodes of the Internet.

Furthermore, progress is being made on ways to better use encryption but in order to really improve privacy we need much more attention to be paid to effective methods for countering traffic analysis. That calls for work at many layers of the stack as private information in web transactions can be exposed via meta-data inferred further down the stack (e.g. via analysis of the sizes of ciphertext packets). While various new protocols (e.g. HTTP/2, TLS1.3) do or will include basic mechanisms that enable mitigations (e.g. padding), it is very unclear how to use those mechanisms in an overall effective manner. Similarly, research here aiming to improve web security and privacy needs to encompass non-web infrastructure protocols such as DNS.

In general, the endeavor to strengthen end-user privacy against tracking and surveillance requires significant innovation in the browser space. This might prove problematic, due to the existing ties of the browser vendors to the advertisement industry (as it is for instance the case with the Google Chrome browser) and the fact that all major browsers are produced in the USA, a major driver of pervasive monitoring. Thus, in consequence, it appears to be paramount to increase the European stake in the web platform's client side, e.g., through a European web browser or similar initiatives.

Part II

Comprehensive Overview on European Web Security

Chapter 7

The State-of-the-Practice in today's Web Software (DS.1)

Some security companies, which fight daily battles against internet attacks, share their knowledge and experiences by publishing regular reports. These reports contain collected data and provide an overview about the attacking landscape, in example, in which way the occurrence of certain attacks changed from one year to the next. In this chapter, data from these security reports is presented to get an impression about the trends of web security attacks of recent years.

7.1 Overview of surveyed vulnerabilities

Before presenting data of the different reports and depicting the trend of web attacks, we want to provide a brief overview about the different vulnerability types that were inspected by most of the reports.

7.1.1 Cross-site Scripting (XSS)

Cross-site Scripting (XSS) is a client-side attack as described in [158]. XSS vulnerabilities enable an attacker to inject malicious Javascript code into the target website. XSS exists in three different forms: Reflective, Persistent and DOM-based XSS.

The first mentioned kind can occur when a server puts data from a request unfiltered into its response. One example is, when the value of URL parameters appear unfiltered in the returned HTML code.

Websites are vulnerable against Persistent XSS attacks when corresponding servers store user input unfiltered, for instance, in a database, and return such user input to visitors as part of the website. The process of this attack is similar to Reflective XSS, but instead affecting only the user who clicked on the link, the malicious code an attacker sent is stored and, under certain circumstances, returned to every visitor of the website, resulting in a probably harmful execution of JavaScript in the victim's browser.

When a website contains a DOM-based XSS vulnerability, an attacker is able to execute malicious JavaScript code without modifying the actual HTTP response, only by interacting with the client code. Hence, the server is usually not able to observe the attack. Such an attack could be performed by tricking the user to click on a link that sends a purchase request to the server on the user's behalf.

7.1.2 Cross-site Request Forgery

With a Cross-site Request Forgery (CSRF) attack, a user can unwillingly be forced to execute an action that targets the vulnerable website. If the user is authenticated on the website the attacker can perform actions such as changing the password, purchasing items or other scenarios, depending on the vulnerability's dimension.(cf. [157])

7.1.3 SQL injection

Attacks aiming on server-side injection vulnerabilities try to execute own code or statements on server side, in example, to take over the control or to read sensible data. SQL injection can be exploited when a server executes SQL queries containing unfiltered user input. In this case, an attacker can send data to the server that enables the attacker to modify the used SQL queries and, as one possibility, fetching data of interest.(cf. [161])

7.1.4 Session Management, Authentication and Authorization

Session management enables a server to associate different requests with the same user, allowing the server to store specific information for the same user between different requests. Session Management vulnerabilities allow an attacker to act as a valid website user, in example by using old session IDs that should have been expired by the server. In an authentication and authorization process, the server checks the identity of the user to verify whether or not the user pretends to be someone else and grants the relevant permissions. When this process is not done correctly by the server, an attacker can abuse such an Authentication & Authorization vulnerability. As a result, the attacker can access parts of the application that should actually be secured. In this way, the attacker can access sensitive data or functionalities that should only be accessible by correct authenticated users. (cf. [160], [154])

7.1.5 Further attacking vectors

By exploiting an Information Leakage vulnerability, an attacker can reveal system data and information that enable a more precise insight into the server and its architecture. Such information can, for instance, contain data about the used database, enabling the attacker to tackle the server more precisely with other methods, such as SQL injection.(cf. [159])

Buffer Errors make it possible to overwrite the memory of the server process and, thus, to execute malicious code or to crash the application.(cf. [155])

Content Spoofing is similar to XSS, since it injects code into the website and alters the code of it. In contrast to XSS, no script code is injected into the website, but HTML code or text to provide wrong content to the user. For example, an HTML form could be injected into the website's code which asks for the user's password that is after submitting sent ot the attacker.(cf. [156])

7.2 Cenzic - Application Vulnerability Trends Report

7.2.1 Description

The company Cenzic, which belongs to Trustwave, provides application security solutions to customers. The data for their reports is collected by the Cenzic Managed Security team. In the following, data and information of their reports from 2013 (cf. [17]) and 2014 (cf. [19]) are exposed. Since the reports review the preceding years, the 2013 and 2014 reports present data collected in 2012 and 2013, respectively.

7.2.2 Data

Cenzic measured 637 security breaches in 2012 and only a few less in 2013, with a total number of 619 counted breaches. The 2013 report states that 99% of applications that were tested in 2012 had at least one serious security vulnerability, while this number shrunk marginally to 96%, as stated in the 2014 report. The median number of vulnerabilities in 2012 was 13, which is a slight decrease from 16 of the previous year. In year 2013 the number was increased by 1, resulting in a median of 14.

With a total of 26%, the 2013 report declares XSS to be the most frequently found vulnerability in applications in 2012. This number rose significantly from 17% in year 2011. In 2013, XSS was on top of the vulnerabilities' list again with a small decrease of 1% to a rate of 25% of all found vulnerabilities, by which XSS was still responsible for one out of four web attacks. Whilst the frequency of XSS vulnerabilities increased compared to 2011, the number of affected applications declined steadily from 70% in 2011 to 61% in 2012 and 60% in 2013. This means that less pages contained an XSS vulnerability, but those which did contained more of them than in previous years.

After a drop by 7% from 23% in 2011 to 16% in 2012, Information Leakage was at 23% again in year 2013, by which it was on position 2 of all found vulnerabilities, closely following XSS. Also the rate of applications that contain such a vulnerability went up in 2013, to a total of 36%. In the previous two years this number was constantly at 17%.

The rate of Session Management Errors fell continuously from 20% in 2011 to 16% in 2012 and down to 13% in 2013. The likelihood to find such a vulnerability in an application declined as well, from 82% to 80% and 79% in 2013. This is still a high number, especially when compared to other vulnerability type from which XSS is the closest with a likelihood of 60%.

The amount of CSRF vulnerabilities increased from 6% in 2011 to 8% in 2012 and dropped again to 6% in 2013. Also the number of applications containing a CSRF vulnerability remained constant at 22% in 2012 and 2013, after it was reduced by 2% from 24% in 2011.

SQL injection seems to follow an upwards trend, due to the fact that vulnerabilities of this kind climbed up from 2% in 2011 to 6% in 2012 and stayed at 7% in 2013. Also the likelihood to find an SQL injection vulnerability in an application rose from 10% in 2010 to 16% in 2012 and by additional 4% to 20% in year 2013. The report suspects that this probably is not because coding standards get worse, but rather because detection tools get better in revealing such vulnerabilities.

Figure 7.1 and figure 7.2 display the previous presented data and makes the respective trends visible. The 2014 report claims that default and weak passwords, misconfigurations, missing security patches and difficulties to allocate head count and budget for detecting and correcting application vulnerabilities are the reasons for the high numbers of vulnerabilities and unfixed issues.

7.3 CVE - Common Vulnerabilities and Exposures

7.3.1 Description

The system Common Vulnerabilities and Exposures (CVE) (cf. [140]) is maintained by the not-for-profit MITRE Corporation, which is funded from the US States Department of Homeland Security. The idea behind CVE is to collect known vulnerabilities of publicly released software in a central spot, which can be referenced by other sources. Such a CVE record contains, among other fields, an Id, a description of the vulnerability, a category the record belongs to, the impact the vulnerability has and a list of references for further information.

To collect information about which kinds of vulnerabilities are most common and to get an insight into the trend of vulnerabilities, we analyzed all published CVE records. The website provides a file containing all records, but with less information compared to the respective record website. Therefor, we analyzed the records in two ways. Firstly, we searched the description of

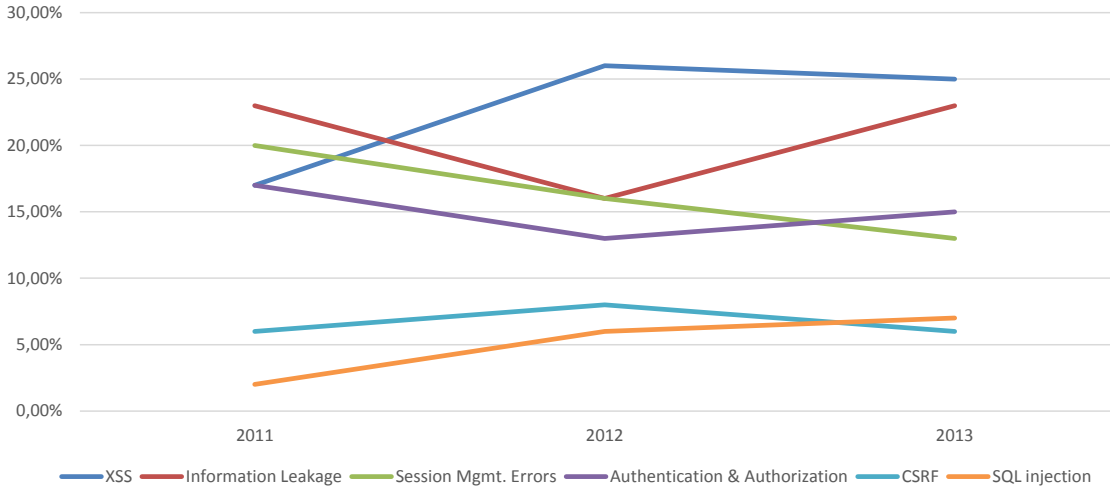


Figure 7.1: Rate of found vulnerabilities [Cenzic]

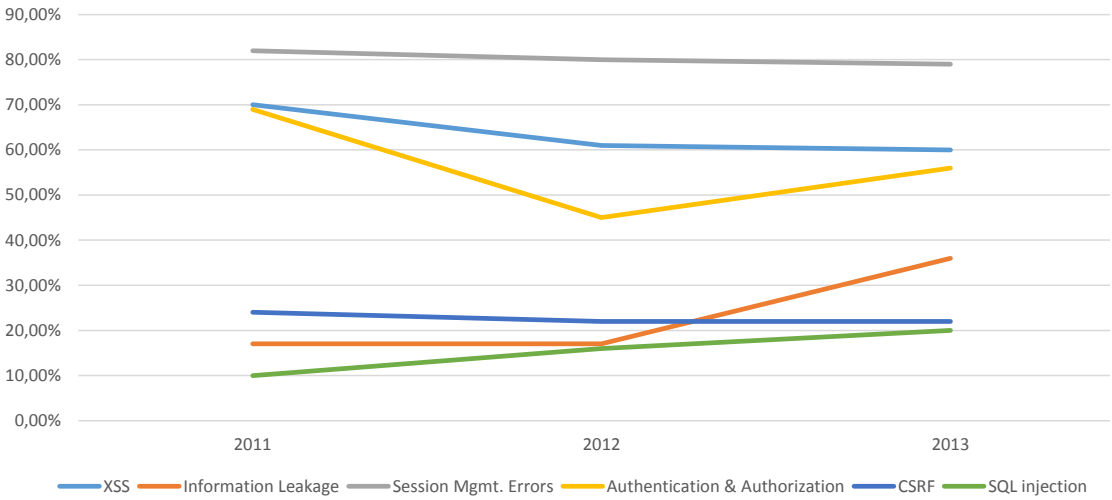


Figure 7.2: Likelihood for website containing vulnerability [Cenzic]

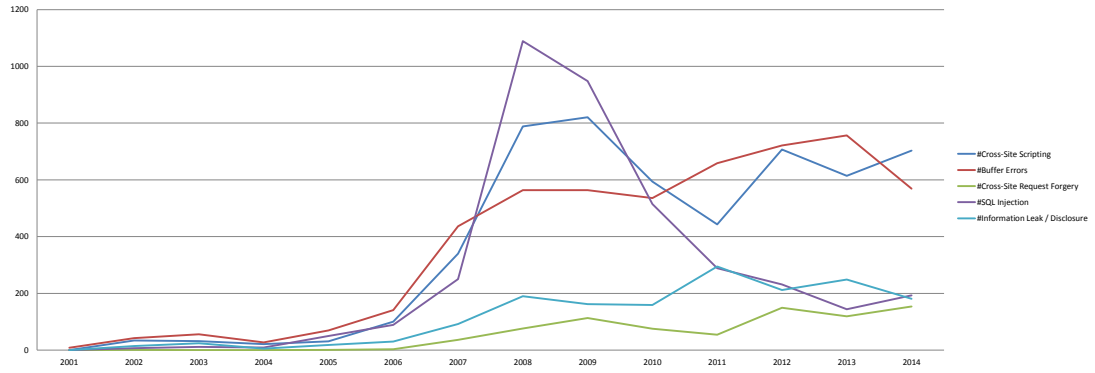


Figure 7.3: Number of CVEs by category [CVE inspection]

each record for certain keywords, indicating which kind of vulnerability it addresses. Secondly, we crawled the website of each record and collected, if available, the year of publication and the category identifier. We counted the number of records that belonged to a specific category and the number of records that contained specific keywords. These numbers are presented in the following.

7.3.2 Data

The data was collected on 09/24/2014 from the CVE website as described in the description of this section. The file with all CVE records contained a total amount of 73.383 CVEs. From this 73.383 CVEs, 8817 had the status 'Reserved' and 679 had the status 'Rejected', leaving 63.887 CVEs that were inspected for their category respective special keywords. Based on the collected data, we created two rankings. One ranking is based on CVEs assigned to a certain category, the other ranking takes the keywords as its foundation. We decided to not only take category information into account but also keywords, because not every CVE record was categorized. We used regular expressions to search descriptions for certain keywords. Since this method does probably not have a precision of 100%, we decided to not only rely on keywords.

The CVE website provides a list with categories (CWE) a CVE can belong to. We checked the category information of each CVE record for being in this list and possibly increased the corresponding counter by one. In case a CVE did not provide any category information, we increased a NoCategory counter. If a CVE had a category that does not have a mapping to a CWE identifier, the category was set to 'Other'.

The number of CVEs categorized as 'Other' is on position 1 by making up 10,40% of all CVEs. 8,18% of the 63887 CVEs were assigned to the XSS category, resulting in XSS being on position 2. XSS was closely followed by CVEs of the Buffer Errors category with 8,07%. SQL injection is in the top five of CVEs ranked by categories, with an occurrence of 5,99%. Information Leakage and Disclosure is on position 9 with 2,57% and CSRF is ranked on position 14 with 1,22%. Figure 7.3 displays the trend from 2001 until today.

We found out that CVE descriptions usually contain the same words when they belong to a certain group of vulnerability. As a result, besides checking to which categories the CVEs belong to, we also searched the descriptions for specific keywords and counted the number of CVEs containing these keywords. The resulting ranking is similar to the outcome of the categorization approach. On position 1 is the group of CVEs that contain Buffer Error related keywords with a total of 14,9% of all CVEs. XSS is with 13,25% of CVEs on position 2, as it is in the category ranking. SQL injection is on position 3 due to 9,34% of CVEs containing SQL injection phrases. On position 6 is CSRF with 1,42%. The trend of CVEs grouped with this keyword approach is displayed in figure 7.4.

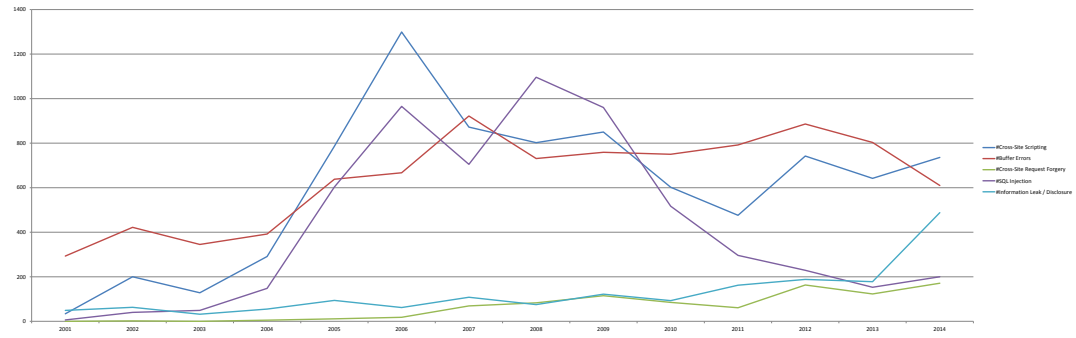


Figure 7.4: Number of CVEs by keywords [CVE inspection]

7.4 Trustwave - Global Security Reports

7.4.1 Description

Trustwave is an American security company that provides on demand security solutions. It was founded in 1995. The data presented in their annual reports are based on data collected during investigations, research and other client engagements. In the following, the data of their reports from the years 2012 (cf. [136]), 2013 (cf. [137]) and 2014 (cf. [138]) are presented. The reports provide insights into the data of preceding years.

7.4.2 Data

The data that is provided in the reports published by Trustwave was collected from many sources that are maintained by Trustwave.

The report from 2014 states that in 2012 99% of all tested websites contained one or more serious security vulnerabilities. In 2013, this situation improved slightly, since the number changed to 96%. Even though the number of websites having at least one security vulnerability were a little bit better, the median number of vulnerabilities became worse by increasing from 13 to 14 in 2013.

The 2012 report created a ranking for web application risks. For this ranking the report took the finding frequency, the launching difficulty, the exploit likelihood and the impact of such an attack into account. Based on these criteria, SQL injection was put on top of the list. XSS reached rank 3 and CSRF obtained rank 7.

The report published in 2013 measured that 82% of tested websites contained an XSS vulnerability, by what XSS had the highest number in this area. Still 72% contained a CSRF flaw and 15% of the websites were vulnerable against SQL injections. Trustwave ranked these vulnerability types based on their risk and their frequency of observation. Although SQL injection was less frequently found than XSS, they declared it to be the number one of application vulnerabilities. The report claims that poor coding practices have allowed SQL injection to remain an attack vector. XSS was ranked 4, right after Miscellaneous Logic Flaws and Insecuire Direct Object Reference. CSRF was put on position 6.

Additionally, Trustwave presents in its report from 2013 the top attack methods referring to the Web Hacking Incident Database (WHID). Based on the WHID data, SQL injection was the second most attack method in 2011 with 23%. Position 1 was the group of all Unknown attacks. CSRF was responsible for 2% of the attacks and XSS only for 1%. Referencing the WHID, the number of CSRF attacks in year 2012 was only 1%, the same number as XSS attacks. SQL injection fell down to 11%.

The 2014 report presents the results of Trustwave's security scanning. Based on this scanning, XSS made up 25% of the found security flaws, closely followed by Information Leakage with 23%,

Authentication and Authorization with 15% and Session Management with 13%. Although not associated with a concrete percentage number, SQL injection is mentioned to be under the top 6 vulnerabilities in terms of frequency and severity.

7.5 WhiteHat Security - Website Security Statistics Reports

7.5.1 Description

WhiteHat Security is an IT security company that was founded in 2001. It is located in California and provides products to support and increase their customer's security in the web. Based on the data collected by their main product, called 'WhiteHat Sentinel', the company started to publish a yearly security report in year 2006. In the following, the data published in the reports from year 2011 to 2013 are presented, see [87], [88] and [89]. Like the Cenzic and Trustwave reports, these reports provide insights into the data collected in the preceded years. The report from 2014 collected information about vulnerabilities based on programming languages, therefor, these numbers cannot be compared to the numbers of previous years.

7.5.2 Data

The reports mention that the average number of vulnerabilities per website plummet from 1111 in 2007 down to 230 in 2010, 79 in 2011 and to its lowest level with 56 flaws in 2012. Although this number decreased heavily, the number of websites containing at least one serious security vulnerability was with 86% still high, as published in the 2013 report.

The trends of the different kinds of vulnerabilities is displayed in figure 7.5. XSS and Information Leakage occupied the first two ranking positions and alternated the top position with one another. In 2010, 64% of the inspected websites contained an XSS vulnerability and the same amount of websites had an Information Leakage issue. In year 2011, the number of websites containing an XSS vulnerability dropped by 9% from 64% to 55%, and the number for Information Leakages dropped even by 11% to 53%. In the following year, 2012, these numbers switched, putting Information Leakage on top of the list with 55% and XSS on position 2 with 53%, hence, each vulnerability remained constant on almost the same level as in the previous year.

In year 2010, XSS and Information Leakage were distantly followed by Content Spoofing, which occurred in 43% of websites. In 2011, 7% less websites had an Spoofing flaw and in 2012 this number was reduced by further 3%, resulting in 33%.

CSRF vulnerabilities appeared in 24% of websites in 2010, which is 40% less than XSS or Information Leakage and around 20% less than Spoofing. This number lessened to 19% in 2011, only to climb up to 26% in 2013.

SQL injection started with 14% in 2010, losing 3% in 2011 to end with 7% in 2012.

Although all mentioned vulnerabilities lost points from 2010 to 2011, we cannot determine a general downwards trend of websites containing security vulnerabilities. After the drop from 2010 to 2011, XSS and Information Leakage stayed almost on the same level. The likelihood of having a Spoofing flaw indeed decreased over the years, but this trend also slowed down from 2011 and 2012. CSRF, in fact, had its all-time high in 2012, although its number first decreased from 2010 to 2011. SQL injection dropped by half from 2010 to 2012, but 7% is still a concerning number for a vulnerability that the security world is aware of for many years now.

7.6 Others

In this section we mention further statistics of other reports that make some interesting statements.

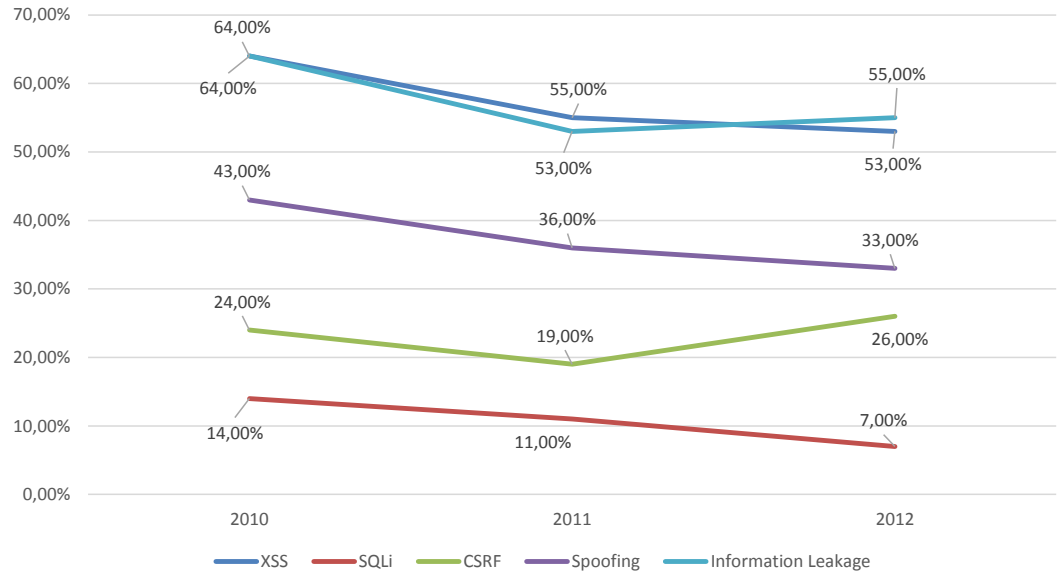


Figure 7.5: Likelihood for website containing vulnerability [WhiteHat]

For Verizon's 2014 Databreach Report (cf. [18]), which reviews preceding years, the company investigated 63437 incidents, around 15000 incidents more than in the previous year. Verizon offers a lot of security data in these reports, however, it is not in their focus to give granular insights into different kinds of web attacks such as XSS, SQL injection or CSRF, but rather a general overview about web attacks. They revealed that the main motivation behind web app attacks was Ideology/Fun with 65% out of 1126 web attacks. Financial purposes were the second motivation with 33% and a small amount with 2% is assigned to Espionage. The report states that the used attacking vectors differ depending on the target branch. In example, in 80% of attacks targeting the retail industry, an SQL injection vulnerability was used, by what it was the main attack vector in this sector. This exemplary number stresses the high relevance SQL injection still has. Unattached to some special industries, Verizon claims that attacks mainly targeted content management systems, such as Joomla! or Wordpress.

Symantec is an IT company that provides security, storage and systems management solutions to support protecting the customer's infrastructure against risks. The data for their annual Internet Security Threat Report 2014 (cf. [20]), which is based on the data of preceding years, was collected with help of their Symantec Global Intelligence Network consisting of sensors. The number of breaches increased by 62% from 2012 to 2013 with 156 and 253 identified breaches, respectively. The number of exposed identities rocketed from 93 mio. to 552 mio. by 493%, which shows how important Internet security is to prevent such incidents. The number of vulnerable websites is denoted to 77% by Symantec's report, which is an increase of 24% compared to 2012. 16% of these vulnerable websites were marked as being critical, enabling attackers to leak relevant data, deface websites or attacking the visitor's computer. It is stated in the report that the number of Zero-day vulnerabilities increased by 64% from 14 in 2012 to 23 in 2013, indicating an upwards trend of new vulnerabilities.

Firehost is a leading provider of secure cloud infrastructure. In its Superfecta report (cf. [67]), Firehost presents numbers of security threats measured by its Intelligent Security Model. XSS, SQL injection and CSRF were accounted to be the most serious threats of businesses. In its Superfecta report that reviews the year 2013, 35% of the measured incidents were XSS attacks, 18% SQL injections and 24% CSRF. The number of XSS attacks decreased by 9% from 44% to 35% from 2012 to 2013 and the number of CSRF lessened from 28% to 24%. SQL injection was

the only vulnerability kind that had a higher frequency with 18% in 2013 than in the previous year with 13%.

7.7 Comparison

The focuses of the annual reports of each publisher can be very different, which is, for instance, the reason why the Whitehat Security Report 2014 was not taken into account. Since even the reports published by the same organization can be so different, it is not always trivial to compare the numbers of completely different reports with each other. However, table 7.1 shows the comparable numbers of the different reports side-by-side, to provide a quick overview. For the mentioned reason, not every cell for each report is filled.

For the Cenzic, TrustWave and WhiteHat columns, the left number indicates the rank based on the likelihood to find such a vulnerability in a website, whereas the right number corresponds to the ranking based on the amount of findings. These numbers can differ, when one website contains more than one security vulnerability of the same kind. In this case, the prevalence increases, whereby the number of websites containing such a vulnerability remains constant. In the CVE column, the first number expresses the ranking by categories and the second number the ranking by keywords.

It is noticeable that XSS takes the top position in each report. The positions of the other vulnerabilities is, at least in one of both groups, also similar. Among other aspects, differences originate from different focuses of the reports, distinct measurement methods and varied data sets. The numbers of vulnerable websites, a number between 77% and 96%, is startling, especially, because well-known vulnerabilities such as SQL injection are still ranked on a high position. The number of web attacks in general fluctuated over the past years and, although the number of some kinds of attacks decreased over the time, we cannot see an obvious improvement. When some kinds of attacks are falling, other ones arises and attacks we thought of being defeated in one year, celebrated their comeback in the following. Hence, we are far away from a secure web and have to spend more resources than ever to face old, well-known and new threats.

Table 7.1: Vulnerability ranking of the different reports by likelihood/prevalence

Vulnerability	**CVE (category/keyword)**	**Cenzic**	**TrustWave**	**WhiteHat**
XSS	2 | 2	2 | 1	1 | 1	2 | 1
SQL injection	4 | 3	7 | 5	6 | /	14 |>5
CSRF	14 | 6	6 | 6	2 | /	5 | 5
Information Leakage	9 | /	5 | 2	/ | 2	1 | 4
Spoofing	/ | 4	/ | /	/ | /	3 | 2
Session Management	/ | 11	1 | 4	/ | 4	8 | /
Authentication & Authorization	5 | /	3 | 3	/ | 3	10 |>5
Other	1 | /	/ | 5	10 | /	/ | 3

Chapter 8

Selected Empirical Studies (DS.2)

While general statistic compiled by third parties, as collected in the previous chapter, are a valuable asset in respect to assessing the overall state of the subject, they do not suffice when it comes to examine more specific aspect of current practices Web security and topical deep dives. For this reason, the STREWS consortium decided to conduct a limited set of empirical studies to explore selected topics.

In this Chapter, we report on three large-scale empirical studies:

- *Assessment of the observable security practices of the European Web:* In the recent years various security mechanism have been added to the Web, some in the form of HTTP response headers, some in the form of other observable protocol artifacts, and some in the form of coding conventions. Wether these techniques are picked up by a given Web application/site can be determined by analysing the site's HTTP responses and the corresponding HTML. In a large scale examination of popular European Web sites, we explored the current level of awareness and adoption of these security technologies in the European Web in September 2013 (Section 8.1).

  In addition, we reassessed the same websites in September 2015, and investigated how the overall web security did evolve over time, and which websites did improve, either by applying the security features more consistently over their domain or by applying new security techniques as part of their defense mechanism. The report of this longitudinal study can be found in Section 8.2.

- *Large-scale analysis on client-side complexity based on DOM-based XSS measurements:* The recent years have brought a significant push of application logic from the server to the client-side. Modern Web applications consist of a considerable amount of JavaScript code that is executed in the user's browser. Only little inside exists on the complexity of this code and the security implications, that are connected with this shift. Hence, as a first measure we conducted a practical study on DOM-based Cross-site Scripting, a sub-class of XSS (see Sec. 7.1.1) that is caused by insecure JavaScript code.

- *Large-scale Analysis of Third-party Security Seals* A third-party security seal is typically an image that can be embedded in a website to signal consumers that the website has been scanned by a security company and has been found to be without security issues. We have performed a large-scale analysis of third-party security seal providers, to assess whether these seals can be trusted and to what extent the seal providers can accurate detect vulnerabilities in a given websites. Moreover, we investigated the existing ecosystem of third-party security seals. The report of this study can be found in Section 8.4.

8.1 Large-scale Security Analysis of the European Web

As the web expands in size and adoption, so does the interest of attackers who seek to exploit web applications and exfiltrate user data. While there is a steady stream of news regarding major breaches and millions of user credentials compromised, it is logical to assume that, over time, the applications of the bigger players of the web are becoming more secure. However, as these applications become resistant to most prevalent attacks, adversaries may be tempted to move to easier, unprotected targets which still hold sensitive user data.

In [200], we report on the state of security of the Web in September 2013 for more than 22,000 websites that originate in 28 EU countries. We first explore the adoption of countermeasures that can be used to defend against common attacks and serve as indicators of "security consciousness". Moreover, we search for the presence of common vulnerabilities and weaknesses and, together with the adoption of defense mechanisms, use our findings to estimate the overall security of these websites. Among other results, we show how a website's popularity relates to the adoption of security defenses and we report on the discovery of three, previously unreported, attack variations that attackers could have used to attack millions of users.

8.1.1 Introduction

Over the last decade, the web has become extremely popular. Businesses heavily depend on the web for their day-to-day operations, and billions of users interact on social networking websites on a daily basis. As a consequence of this enormous growth in popularity, the web has also drawn increased attention from attackers. A whole range of web attacks exists in the wild, ranging from Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and SQL injection, to the exploitation of broken authorization and session management. Moreover, as the technologies that support the web increase in numbers and complexity, new opportunities for exploitable vulnerabilities increase with them.

To assess a website's security, website owners typically choose security consulting firms for internal penetration testing, and code reviewing. It is difficult, however, for outsiders like government and supervisory organizations to assess a website's security externally, especially when the assessment needs to be done at a larger scale, e.g., involving a large number of websites belonging to a country, or a specific industry sector. Such an assessment may be desirable since the citizens of each country depend more and more on certain web applications for their daily lives. An example of a real-world equivalent is the mandatory assessment of the structural safety of buildings in order to protect people from future disasters that could have been straightforwardly avoided.

In this chapter, we investigate the feasibility of external security evaluations through a large-scale security analysis of the web. In particular, we evaluate the security stance of popular websites in the European Union (EU), and investigate the differences among countries.

To evaluate a website's security, existing approaches typically focus on the discovery of vulnerabilities in websites. For example, WhiteHat publishes yearly reports on website security statistics [218], highlighting the ten most common vulnerabilities, and discussing new attack vectors. Contrastingly, our approach not only accounts for common vulnerabilities and weaknesses, but also measures the presence of security mechanisms deployed on the investigated websites. These mechanisms have been developed by the security community as a response to web application attacks, making their adoption a crucial step towards a more secure web. The presence or absence of each of these mechanisms can be passively detected and can be used as an indicator of the "security consciousness" of each individual site.

In addition, in order to be able to compare websites by their security posture, we also propose a security scoring system for assessing a website's security level, and based on the scoring system, we present a comparative security analysis of European websites. Finally, because of the breadth of our analysis, we report on the discovery of novel variations of existing web application attacks. In one of the discovered cases, an attacker can register an expired Google Code project and serve

malicious JavaScript to millions of users of sites that once trusted that specific project for remote code.

Our findings allow the community to assess the adoption of security mechanisms by websites at a large scale, and also prioritize corrective action, based on the severity of the discovered issues. Moreover, we list the challenges that we faced in our experiment, and provide possible directions towards future research in the area.

8.1.2 Data Collection

Dataset

For our experiment, we selected popular websites from the EU as the targets, to evaluate website security, and investigate the presence of potential differences between countries. The 28 member states in the EU represent a diverse set of communities, each with their own demographic characteristics. For each EU country, we selected the top 1,000 websites ending with a country code top-level domain (ccTLD) from Alexa's list of the top 1 million sites. For example, 1,000 websites ending with the Belgian ccTLD '`.be`' are extracted to represent the Belgian web. Note that several small EU countries (such as Luxembourg and Malta) do not have 1,000 websites in Alexa's top 1 million list, so we end up with a few countries having less than 1,000 websites in our dataset. We then obtained up to 200 webpage URLs for each website by querying the Bing search engine [2] for the popular webpages of each website. In total, we analyzed more than 3 million webpages for 22,851 EU websites with an average of 141 webpages per website.

Crawler setup

After the webpage URLs are obtained, PhantomJS [10], a headless browser, is used to visit the URLs and retrieve data from webpages. By loading every webpage within PhantomJS, we mimicked the behavior of a regular visitor using a Chrome browser. In order to crawl a large number of webpages in reasonable time, we run the experiment in a distributed fashion using 60 networked machines. As a result, our crawling experiment took approximately five days.

8.1.3 Security Scoring System

In order to compare the security level among different websites, and among different EU countries (represented by the websites of each country), we developed a security scoring system that gives quantitative security scores for each website. The security scores for a website consist of two parts: a positive score to represent the defense mechanisms adopted by the website (such as the `X-Frame-Options` and `Content-Security-Policy` headers), and a negative score for vulnerabilities or weaknesses (such as vulnerable remote JavaScript inclusions and insecure SSL implementations) found on it. For each defense mechanism and vulnerability/weakness, the security scoring system assigns a weighted positive and negative score. The overall positive and negative score for a website, is obtained by summing up each weighted positive and negative score respectively.

Due to our ethically-guided choice of conducting passive analysis for the majority of our tests, our search was limited to eight defense mechanisms, and ten vulnerabilities/weaknesses for each website. In principle, however, the security scoring system is scalable to more measurements. In the following sections, we briefly describe these defense mechanisms and vulnerabilities/weaknesses and elaborate on the scoring system we adopted.

Defense mechanisms

In our security assessment for defense mechanisms, we searched whether each website had adopted one or more of the following eight mechanisms:

- **HTTP Strict-Transport-Security (HSTS):** HSTS is a web security policy mechanism where a web server can force complying browsers to interact with it using only HTTPS connections [101]. By sending out the HSTS policy via an HTTP response header named `Strict-Transport-Security`, a web server specifies a period of time during which the user's browser is instructed that all requests to that website need to be sent over HTTPS, regardless of what a user requests. As a result, HSTS can effectively thwart SSL-striping attacks and other Man-in-the-Middle (MitM) attacks [132, 151].

- **Secure Cookies:** Website operators can make use of the `Secure` flag when sending out `Set-Cookie` headers. By doing so, the scope of a cookie is limited to only secure channels [32], which makes the cookie less likely to be stolen via eavesdropping.

- **Content Security Policy (CSP):** To mitigate a wide range of injection vulnerabilities, such as Cross-Site Scripting (XSS), a website operator can make use of the CSP mechanism. CSP provides a standard HTTP header that allows website owners to declare approved sources of content that browsers should be allowed to load on any given webpage [188]. Whenever a requested resource originates from a source that is not defined in the policy, it will not be loaded [190]. Hence, if the policy does not allow in-line JavaScript, then even if an attacker is able to inject malicious JavaScript in the webpage, the code will not be executed.

- **HttpOnly Cookies:** By default, cookies are accessible to JavaScript code, which allows attackers to steal a user's cookies in an XSS attack. To protect against the theft of cookies, a website operator can use the `HttpOnly` flag on cookies. An `HttpOnly` cookie will be used only when transmitting HTTP/HTTPS requests, making them unavailable to client-side JavaScript.

- **X-Frame-Options (XFO):** When an attacker is able to load a website, or part of a website in a `frame` or `iframe` element, the website might be vulnerable to ClickJacking attacks. More precisely, by redressing the user interface, an attacker can trick the user into clicking on the framed page while the click is intended for the bottom-level page [128]. To avoid ClickJacking attacks, the XFO HTTP response header [176] can be used to instruct a user's browser whether a certain page is allowed to be embedded in a frame.

- **Iframe sandboxing:** The `sandbox` attribute for the `iframe` element, introduced in HTML5, enables a set of extra restrictions on any content loaded in a frame. By specifying the `sandbox` value, a website operator can instruct the browser to load a specific frame's content in a low-privilege environment, allowing only a limited subset of capabilities to be made available to that frame [212].

- **CSRF Tokens:** The most popular defense for Cross-Site Request Forgery (CSRF) attacks is the inclusion of a secret token with each request and validation of that token at the server side [34]. This secret token, often referred to as a "nonce", should be pseudo-random and of a certain length so it cannot be guessed or brute-forced by an attacker. To check for nonces, we searched for forms that contained a hidden form element that was most likely used as a nonce. More specifically, form elements were marked as nonces when their name contained the keywords "token", "nonce", or "csrf", and when their value was a long alpha-numerical string. These form elements were then manually verified in order to filter out any false positives.

- **X-Content-Type-Options:** Internet Explorer has a MIME-sniffing feature that will attempt to determine the content type for each downloaded resource. This feature, however, can lead to security problems for servers hosting untrusted content. To prevent Internet Explorer from MIME-sniffing, thus reducing exposure to attacks, a web server can send the `X-Content-Type-Options` response header with the `nosniff` value .

Apart from the sandboxing of frames and CSRF tokens, all the above defense mechanisms are communicated to the browser via HTTP response headers, and hence can be discovered straightforwardly by parsing a server's response headers. For the sandboxing of frames and the presence of CSRF tokens, we searched for `iframe` and `form` elements in the response body of each crawled webpage.

Vulnerabilities and Weaknesses

For the assessment of vulnerabilities and weaknesses, we focus on the following ten measurements:

- **Vulnerable Remote JavaScript Inclusion:** A website that chooses to include JavaScript from untrustworthy third-party sources opens itself up to a range of security issues. Recent research by Nikiforakis et al. [149], identified four different types of vulnerabilities that are related to the practice of unsafe remote JavaScript inclusions. In our assessment, we searched for the most dangerous of these vulnerabilities, called "Stale Domain-name-based Inclusions", where remote JavaScript is requested from a domain that has expired and is available for registration, which means the attacker can buy the domain and use it to serve malicious JavaScript.
- **Mixed-content Inclusion:** When migrating to HTTPS, many websites fail to fully update their applications, resulting in mixed-content inclusions where the main webpage is sent over a secure HTTPS channel, while some additional content included on that page, such as images and scripts, are delivered over non-secured HTTP connections. As a result, an active network attacker can attack the TLS-enabled website by intercepting and modifying any of the mixed content that is loaded over HTTP [51].
- **SSL-stripping Vulnerable Form:** For performance reasons, some websites only implement HTTPS for certain webpages that contain sensitive information (such as a log-in page), which may result in forms vulnerable to SSL stripping [132]. In this scenario, the form is displayed on an HTTP page, however the form action points to an HTTPS link. As a result, a MitM attacker can replace all HTTPS form links on the HTTP page to HTTP links, which will allow the attacker to intercept the form data sent from the user's browser.
- **Insecure SSL Implementation:** SSL is important for website owners since it provides end-to-end security. At the same time, however, it turns out that it is not easy to deploy SSL correctly. According to Qualys' latest SSL survey of the most popular websites in December 2013, about half of the HTTPS websites have security issues associated with their SSL implementations [12]. In our assessment, we use a fast SSL scanner called `sslyze` [13] to search for SSL implementation issues including the support of SSL v2.0, use of weak ciphers, and the vulnerability to the recently discovered BEAST [69] and CRIME [173] attacks.
- **Weak Browser XSS Protection:** Most modern browsers include security mechanisms to protect a user against reflected Cross-Site Scripting attacks [139], and these features are, in general, enabled by default. While web servers can instruct a user's browser to disable this protection by means of the `X-XSS-Protection` response header, we consider such behavior as a weakness of the website, because disabling it might allow an attacker to successfully exploit an, otherwise unexploitable, XSS vulnerability.
- **HTTP Parameter Pollution (HPP):** When a website fails to properly sanitize user input, they might be vulnerable to HPP attacks. These attacks consist of injecting encoded query string delimiters into other existing parameters. By doing so, an attacker is able to compromise the application logic to perform client-side and server-side attacks. In our assessment, we searched for HPP vulnerabilities in a manner similar to the methodology of Balduzzi et al. [29].

- **Outdated Server Software:** It is important to keep web servers up-to-date, since an outdated server usually contains vulnerabilities that may lead to attacks. In our assessment, we searched for outdated server software for popular web servers including Apache, Microsoft-IIS, and Nginx.
- **Outdated Content Manage Systems (CMSs):** Many popular websites nowadays are built using a CMS, since CMSs allow non-technical users to build dynamic websites, and are usually free of charge. Similar to web servers, it is also recommended to keep a CMS up-to-date, as outdated CMSs often contain vulnerabilities. In our assessment, we looked for outdated CMSs for websites using WordPress, Joomla, vBulletin, and MediaWiki.
- **Information Leakage:** Many websites generate error messages and display them to users, which may reveal implementation details or information that is useful to an attacker. In our assessment, we searched for various categories of information leakage including SQL error messages, website directory listings, IIS error messages, PHP/ASP/JSP source code and error messages.
- **Sensitive Files:** A website may accidentally expose sensitive files such as configuration files and source code to the public, when moving files from the development server to the production server. The degree of vulnerability depends on the sensitive file that is exposed, ranging from information disclosure, to disclosure of source code containing credentials. In our assessment, we searched for the following files that were most likely to contain sensitive information: `phpinfo.php` or `test.php`, containing system information from the `phpinfo()` function, website configuration files, such as `Web.config`, and two version control system folders, namely `.svn/` and `.git/`.

Most of the aforementioned vulnerabilities and weaknesses can be discovered through passive analysis with PhantomJS visiting webpages, except for the finding of HPP vulnerabilities and sensitive files, where we actively scanned a limited number of webpages from each website, taking the necessary precautions not to stress or harm websites.

Scoring System Details

The scoring system used to estimate the state of security of websites is based on the Common Weakness Scoring System (CWSS) [4]. The CWSS provides a quantitative measurement of security weaknesses in software applications, and is mainly used to prioritize the remediation of reported weaknesses. In essence, the score appointed to a weakness by the CWSS aims to reflect the impact and likelihood of exploitation by adversaries. For instance, not "escaping" user-controlled data in an HTML document could lead to Cross-Site Scripting attacks, and may allow an attacker to extract sensitive user information. This weakness obviously has a high impact and is remotely exploitable; thus it will receive a higher score than, say, an insecure SSL implementation weakness.

The reason for using the CWSS over other scoring systems as a base for our scoring system is twofold. First, the CWSS is a well-established and commonly used mechanism to give a quantitative score to weaknesses. It has been extensively reviewed, which gives, to a certain extent, a guarantee that the score appointed to a weakness reflects the magnitude of the induced threat. Second, the CWSS gives scores to weaknesses, rather than to actual vulnerabilities as is done in the Common Vulnerability Scoring System (CVSS) [3]. This is important because most features we analyzed are security indicators rather than actual vulnerabilities.

The CWSS uses 18 different factors across three metric groups to calculate the total score for a weakness. The first group, named the "Base Finding" group, reflects the risk of the weakness, the finding confidence and the presence of built-in defense mechanisms. The second group, called the "Attack Surface" group, reflects the exploitability of a weakness. A vulnerability which is easy to exploit, such as a stale JavaScript inclusion, will consequently receive a higher score for

this group. The last group, named the "Environmental" group, indicates, among others, the impact on the business in case the weakness is exploited, as well as the likelihood of discovery and exploitation. Each group is appointed a subscore which constitutes of a weighted score of its factors. The total score appointed to a weakness is calculated by multiplying the score for the "Base Finding" group (value between 0 and 100) by the two other groups (values between 0 and 1).

In order to give a metric to security features on a similar scale as weaknesses, the CWSS was also used to appoint scores to these defense mechanisms. As the CWSS only works for weaknesses, we calculated the score for security measures by determining the metric for the vulnerability or weaknesses they attempt to prevent. Additionally, we took the effectiveness of the countermeasure into account, as security features that completely block certain attacks should receive a better score. For instance, the `HttpOnly` flag on cookies may prevent sensitive cookies to be stolen in Cross-Site Scripting attacks, but it will not mitigate all consequences of these attacks, something that a properly written Content Security Policy may do.

Table 8.1 shows the score appointed to each defense mechanism and weakness. Due to reasons of brevity, we limit the discussion of the rationale for the calculated scores to one example. As can be seen in the table, the vulnerable inclusion of remote JavaScript received the highest score (67.50). The high impact, i.e., the execution of arbitrary JavaScript code on multiple web pages, and the ease of exploitability, i.e., the registration of a stale domain name, are the main factors that contribute to this high score. Additionally, no control mechanisms (e.g. Content Security Policy) were found on the vulnerable websites that attempt to mitigate this vulnerability. Consequently, a score of 90 was calculated for the "Base Finding" group subscore. As victims will be exploited upon visiting the vulnerable web site, a score of 1 was appointed to the "Attack Surface" group. The score for the "Environmental" group is 0.75. The main factor that contributed to this score is the business impact, which is mostly case-specific. While the execution of arbitrary JavaScript code may have a very high impact on security-sensitive websites (e.g. a banking website), the potential impact on a purely informational website that stores no sensitive data is considerably less. The total score, 67.50, is then calculated by multiplying the three subscores (90 * 1 * 0.75).

8.1.4 Findings

General findings

Out of the 22,851 analyzed websites, we found that 10,539 (46.12%) enabled at least one security feature. As can be seen in Table 8.2, the most popular defense mechanism is the `HttpOnly` attribute on cookies, which was present in 33.51% of the evaluated websites. This defense mechanism is followed in popularity by the presence of a CSRF token in forms, which was found in 16.70% of the websites. Interestingly, these two most popular security features are mitigations for the most critical web application flaws according to the OWASP Top 10 project [9]. This table also shows that, in general, the popularity of a defense mechanism is related to the time it was adopted by popular browsers, i.e., the older a security feature, the more widely it is used.

In our evaluation, we found that 12,885 (56.39%) websites contained at least one vulnerability or weakness. Table 8.3 shows the distribution of the number of websites found to be vulnerable. While only 5,113 websites provided at least one page over HTTPS, we found that the majority (80.32%) had content originating from an insecure channel on their website, or had SSL implementation issues. Likewise, although we only evaluated 17,910 websites for the presence of HTTP Parameter Pollution (HPP) vulnerabilities, we found 15.24% of these websites to be vulnerable. As HPP is very closely related to XSS in the sense that they are both caused by improper encoding of certain characters, we manually analyzed a subset of the webpages vulnerable to HPP for XSS vulnerabilities. This showed us that approximately 75% of the websites vulnerable to HPP are also vulnerable Cross-Site Scripting attacks.

Table 8.1: Calculated scores for defense mechanisms and vulnerabilities

Defense Mechanism	**Score**
Content Security Policy	58.93
`X-Frame-Options`	45.21
HTTP Strict-Transport-Security	33.52
CSRF tokens	32.73
Secure cookies	31.84
HttpOnly cookies	28.21
Iframe sandboxing	25.32
`X-Content-Type-Options`	8.02

Vulnerabilities and Weaknesses	**Score**
Vulnerable remote JavaScript inclusion	67.50
Sensitive files	41.81
SSL-stripping Vulnerable Form	30.16
`X-XSS-Protection`	28.33
Outdated CMS	18.30
Insecure SSL implementation	18.10
HTTP Parameter Pollution	18.06
Mixed-content inclusions	13.42
Information leakage	9.44
Outdated Server Software	8.71

Table 8.2: Results from the analyzed websites that enable security features

Security mechanism	**# of websites**	**% of websites**	**Estimated year of adoption**
`HttpOnly` cookie	7,658	33.51	2007
CSRF token	3,815	16.70	NA
Secure cookies	1,217	5.33	2007
`X-Frame-Options`	1,029	4.50	2008
`X-Content-Type-Options`	467	2.04	2008
`Strict-Transport-Security`	116	0.50	2010
Content Security Policy	13	0.06	2011
Iframe sandboxing	10	0.04	2010

Incorrect security-header usage

By making use of headers, website administrators are capable of instructing a user's browser to enable a certain security feature. Browsers, however, require the value of the security-header to be correct. Values that are incorrect, for example headers containing a typing error or headers with incorrect syntax, will be ignored by the browser. The presence of such headers in websites is a strong indication that the website administrator is under the impression that he successfully secured his website. Nonetheless, if the security-header contains a syntactical or typographical error, adversaries might be able to successfully exploit a vulnerability on a website.

In our analysis, we found several instances where the website operator tried to protect his website against ClickJacking attacks by using the `X-Frame-Options` header, but failed to do so by using an incorrect directive, for instance specifying `SAME-ORIGIN`, instead of the correct `SAMEORIGIN` directive.

Additionally, we found that 15 out of 116 (12.93%) analyzed websites that make use of the `Strict-Transport-Security` header to prevent SSL-stripping attacks, used the header in an improper fashion. The majority of these websites sent the `Strict-Transport-Security` header over an HTTP connection, without referring the user to an SSL-connection. Since browsers will

Table 8.3: Results from the analyzed websites that contain vulnerabilities

Vulnerability	**# of websites**	**% of websites**
Outdated server Software	6,412	28.06
Mixed-content inclusion	3,442	15.06
SSL-stripping Vulnerable Form	2,884	12.62
HTTP Parameter Pollution	2,731	11.95
Outdated CMS	2,041	8.93
Insecure SSL implementation	1,945	8.51
Information leakage	1,231	5.39
Sensitive files	1,068	4.67
Vulnerable remote JS inclusion	91	0.40
`X-XSS-Protection`	91	0.40

ignore HSTS headers that are sent over an unencrypted channel, users of these websites can still fall victim to SSL-stripping attacks. The remainder of websites that implemented HSTS incorrectly, either forgot the `max-age` directive, or set this directive to the value `0`, which signals the user's browser to delete the HSTS policy associated with the website.

Security by Alexa rank

As the set of evaluated websites is distributed over the Alexa's list of the top 1 million websites, we evaluated how the rank of a website relates to the score we appoint it. We found that on average, the rank of a website is positively correlated with the positive score we appoint it, i.e., a high-ranked website is more likely to have a relatively high positive score. Contrastingly, we found that the negative score of a website is unrelated to its popularity according to Alexa. This indicates that popular websites try to improve their security by the adoption of defense mechanisms, rather than by tackling vulnerabilities. Figure 8.1 depicts the relation between the security score and the Alexa rank. Each entry coincides with the average positive or negative metric of the evaluated websites that fall within a range of 10,000 Alexa ranks.

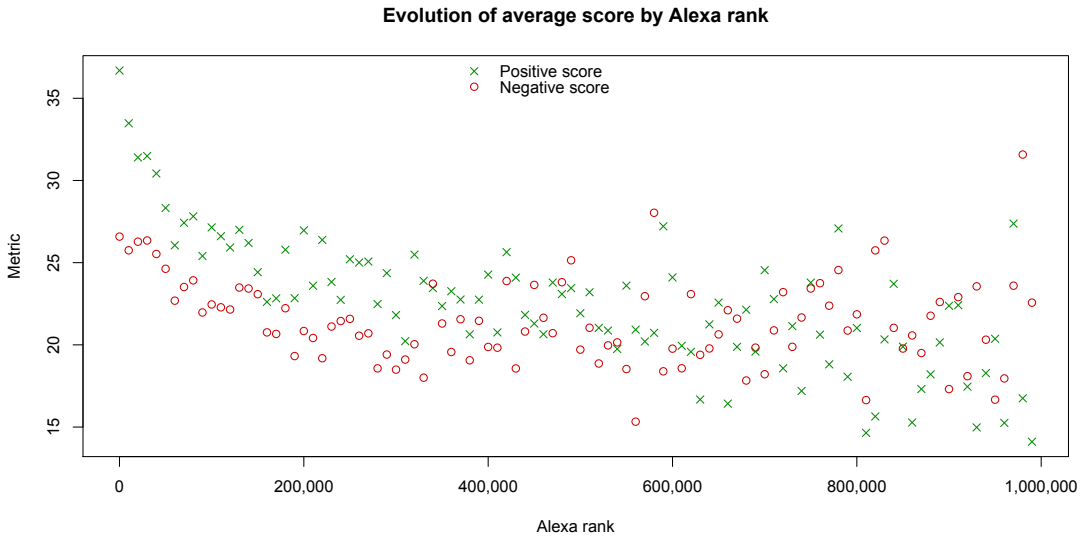


Figure 8.1: Average metric by analyzed websites grouped by 10,000 Alexa entries

Additionally, we found that, in general, there is no correlation between the positive metric and negative score of a website, which strengthens the indication that websites try to improve

their security by adding security mechanisms in an ad-hoc fashion. Moreover, we found that a large number of websites apply a certain defense mechanism to a limited fraction of the URLs we visited, e.g. the majority of websites that make use of the `X-Content-Type-Options` header to prevent XSS attacks in Internet Explorer due to MIME sniffing, only add the header to a small fraction of their pages.

Security by country

We found that the scores for websites located in different countries were similar. Figure 8.2 shows the cumulative distribution function of both the positive as well as the negative score for websites of a set of four randomly selected countries. From this set, Germany has more websites with a higher positive score than the other countries. However, the same country scores worse than the rest on the negative score. This again shows that there is no relation between the number of enabled security features and the number of weaknesses or vulnerabilities we were able to find on a website.

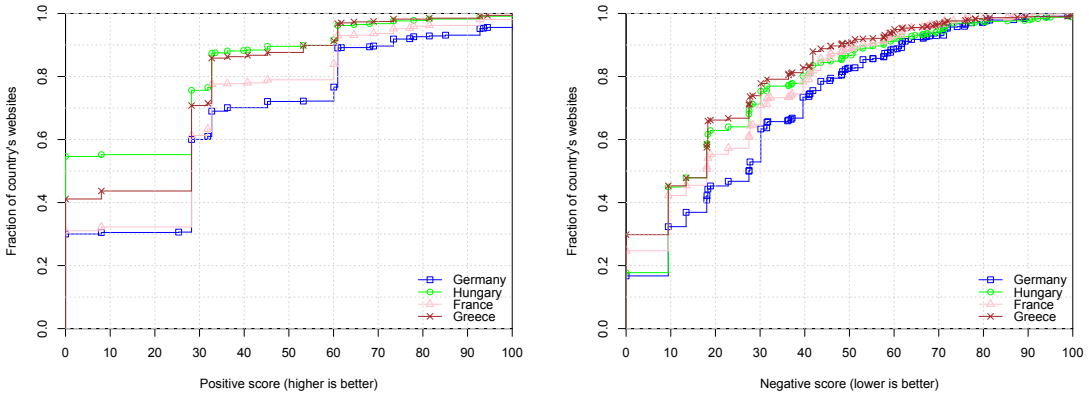


Figure 8.2: Distribution of positive and negative score for several countries' websites

The variance of scores between different countries are most likely due to the unequal distribution of the countries' websites over the Alexa rank. The distribution of Alexa rank for the subset of four countries is shown in Figure 8.3. Compared to the distribution of the positive score, it is clear that the countries with the most high-ranked websites have a better positive score. This indicates that, in general, the security of a website is unrelated to its geographical location or the policies its hosting country may have.

Novel attack techniques

In the course of our analysis, we encountered a new attack technique in the Cross-Origin Resource Sharing mechanism as well as two variations on the insecure inclusion of remote JavaScript code on a webpage. Both are related to remote trust relations in the sense that the website operator trusts a certain domain or URL to be benign, which may become malicious in the future.

By sending out the `Access-Control-Allow-Origin` header, a website operator can instruct a browser to allow a third-party website to make XHR-requests towards his website and read out the result. When the `Access-Control-Allow-Credentials` header is included as well, these requests can be authenticated. It is in the best interest of a website administrator to only allow trusted websites to extract the response of an XHR-request targeting his website. Interestingly, we found a case where a website sent out the `Access-Control-Allow-Origin` containing a

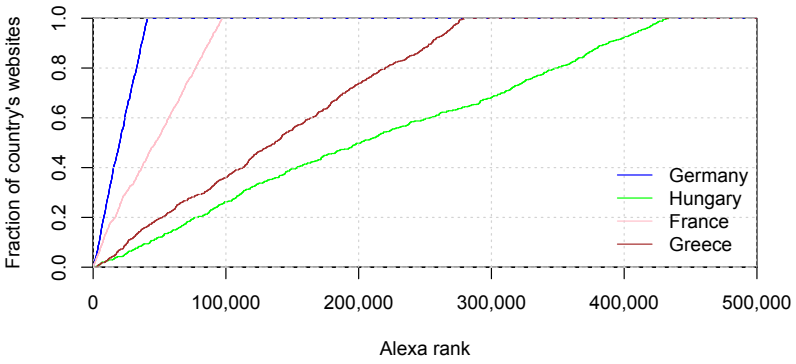


Figure 8.3: Distribution of Alexa rank for several countries

`.local` domain. This allows an attacker to trick a user on the local network in visiting his webpage located at the `.local` domain. The attacker is then able to make the victim's browser send XHR-requests to the vulnerable website while being able to read out CSRF-tokens from forms.

In the aforementioned work by Nikiforakis et al. [149], the authors analyzed the inclusion of JavaScript files from expired domains. In the course of our analysis, we encountered two variations on this type of attack. More specifically, we found that several websites remotely include JavaScript files from domains that were marked as "for sale" by their owner on `sedo.com`, a large domain marketplace. Similar to the attack described by the authors, an attacker is able to buy such a domain, and serve malicious JavaScript to unsuspecting users. The second variation on this type of attack occurs when websites include JavaScript files directly from project hosting websites, such as GitHub or Google Code. The files hosted on these services are linked to a project or a user. However, upon deletion, that project or account, becomes again available for registration. This way, an adversary is able to host malicious JavaScript, which may be included by a large set of websites. To show the importance of this type of attack, we registered a stale project on Google Code, and made available the last available version of project's JavaScript files, with a minor addition which allowed us to analyze the number of including websites and affected users. During a month's time, we registered a total of 3,879,701 requests, originating from 1,104,497 unique IP addresses. In total, there were 3,400 websites, including a prominent Chinese news website, which directly included JavaScript files from this Google Code project. In every single one of these requests, an attacker could have have served malicious JavaScript that steals a user's cookies, exfiltrates private user information and even attempts to launch a drive-by download.

Miscellaneous

In our analysis, we found that the presence of certain security features, such as the `HttpOnly` attribute in the `Set-Cookie` header, is more common in websites that are powered by frameworks which facilitate the system-wide usage of these security features [182]. More precisely, through the `X-Powered-By` header we found that although the majority (49.53%) of the analyzed websites are powered by PHP, only 16.36% of these websites enable the `HttpOnly` attribute. The second most popular framework is ASP.NET, used by 22.80% of the crawled websites. Interestingly, we found that 54.74% of these ASP.NET websites enable `HttpOnly` (three times as much as PHP sites).

8.1.5 Limitation and Challenges

Accuracy of passive analysis

Due to legal and ethical considerations, our analysis of vulnerabilities in websites was limited to a passive analysis, with a few exceptions. Consequently, the results described in the previous section only show an estimation on the state of security of European websites. In order to assess the accuracy of these estimates, we compared the scores of websites likely to be insecure, to websites expected to be secure. The set of most-likely vulnerable websites consisted of websites with a Cross-Site Scripting vulnerability which was publicly known and had not been patched in over one year.[1] The set of websites probable to be secure, was made up from a set of 20 respectable banking websites. This comparison showed that the average positive score for the known-vulnerable set (35.21) was lower than that of the set of banking websites (41.62). Also the average negative score, which was 27.22 on the insecure set and 12.80 on the probably secure websites, indicates that, despite the fact that only a fraction of a website's state of security could be assessed, we were still able to differentiate between vulnerable and secure websites. At the same time, we are aware of the coarse-granularity of our analysis and we highlight the antithesis between the invasiveness of an external security assessment, and the coverage obtained by it. It would be worthwhile to investigate whether website administrators would be willing to consent to a more invasive security assessment, in return for obtaining the results free of charge.

Scoring system

In order to evaluate the general state of security of a website, we developed a scoring system based on CWSS, as was described in Section 8.1.3. However, this scoring system is subject to two types of limitations. Firstly, the total positive score assigned to a website originates from an individually assigned score of eight security features, while the total negative score is derived from a score attributed to ten potential weaknesses and vulnerabilities. As a result, the positive score for a website is on a different scale from the negative score. This prevents us from being able to compare the positive score, to the negative score. Moreover, as the total score appointed to a website originates from a limited set of factors, the total score may not always reflect the actual state of security of a website. However, as we evaluate diverse aspects which are highly related to a website's security, we believe that our scoring system provides a good estimate on the general state of security of a website.

The metric appointed to each weakness and security measure is derived from a list of 18 factors, some of which are subject to the opinion of the authors or are often case-specific. For instance, the impact of exploiting a certain vulnerability may differ based on the type of website. In order to account for these differences, each metric was calculated for a general website. Consequently, the appointed metrics are not website-specific and the score of one feature is relative to the other scores. This allows us to appoint a comparable score which gives an estimation of the state of security for each tested website.

8.1.6 Other studies

To the best of our knowledge, there exists no large-scale analysis which evaluates security features as well as weaknesses in a broad range of websites. Nonetheless, several evaluations on the presence of specific vulnerabilities in web applications have been carried out. For instance, WhiteHat Security evaluates, on a yearly basis, the data on several types of vulnerabilities they collect from their customers [218]. Contrastingly to our research, their security analysis has the permission of their clients and is thus more aggressive, which enables them to find additional types of vulnerabilities, such as SQL Injections and XSS errors.

Another large-scale evaluation of websites in a specific demographic area is presented in research by Alarifi et al. [25], that evaluates the security of popular Arabic websites. Their

[1] `http://www.xssed.com`

analysis explores the presence of phishing and malware pages in 7,000 domains. To detect malicious scripts hosted on webpages, they make use of APIs offered by known website scanners. Kals et al. developed the SecuBat tool, which was used for an automated detection of XSS and SQL Injection vulnerabilities in a selection of 100 security-sensitive websites [114]. Similarly, Zeller et al. performed an analysis on the presence of CSRF vulnerabilities in popular websites [223], finding vulnerabilities in four major websites. Nikiforakis et al. presented a large-scale analysis of remote JavaScript inclusions [149]. Additionally, in their paper, they also proposed a metric called Quality of Maintenance (QoM) to characterize a website's security consciousness. Their QoM adopts several features such as `HttpOnly` cookies, `X-Frame-Options`, that are also included in our assessment. As earlier discussed, the presence of these defensive mechanisms give an indication for a website's security.

Vasek and Moore found that some website features, such as server software and CMSs, can serve as positive risk factors for webserver compromise [203]. Their study shows that some server types and CMS types are more risky than others (e.g., servers running Apache and Ngnix are more likely to be compromised than those running Microsoft IIS).

Lekies et al. performed a large-scale detection of DOM-based XSS vulnerabilities in the top 5,000 Alexa websites [126]. In their evaluation, they found a total of 6,167 unique vulnerabilities distributed over 480 domains, demonstrating that 9.6% of the evaluated websites are vulnerable to this type of attack. Son et al. analyzed the implementation of the HTML5 `postMessage` mechanism in the Alexa top 10,000 [187]. They found that 84 popular websites were exploitable to several attacks, including XSS and content injection, due to the lack of proper checks in the cross-origin communication mechanism.

A feature that we did not include in our study was the security of a site's hosting provider. Sites situated on shared hosting environments are expected to be at a greater risk of compromise, since a vulnerability of another co-located tenant can be used to attack the entire server. Canali et al. recently investigated the ability of shared hosting providers to detect compromised sites hosted on their servers [47], finding that the vast majority of providers cannot detect even the most straightforward attacks.

8.1.7 Conclusion

Websites have become the main target for numerous attacks originating from adversaries who attempt to monetize a user's sensitive data and resources. In order to protect themselves from this threat, website operators are provided with several security mechanisms to defend against a wide range of vulnerabilities. In this section, we evaluated the usage of security features in September 2013, as well as the presence of vulnerabilities and weaknesses, in 22,851 EU websites. We found that a large part of the evaluated websites showed weaknesses, and some even contained severe vulnerabilities. Moreover, we discovered that the state of security of a website is unrelated to its demographic characteristics. In spite the fact that popular websites are more likely to prevent attacks by implementing security features, we found that the presence of weaknesses and vulnerabilities is unrelated to a site's popularity. We hope that our study can inspire similar systems at a country- or sector-level, and help the owners of sites to discover and prioritize the adoption of security mechanisms, and the correction of existing vulnerabilities.

8.2 Evolution of the Security State of the European Web

8.2.1 Introduction

The web is constantly evolving, with new technologies such as HTML5 and CSS3 getting widely used and supported, which provide internet users richer experience. In the mean time, the attacks on the web are also changing, shifting from server exploitation such as SQL injection to client-side attacks such as XSS and Man-in-the-Middle (MitM) attacks such as SSL-stripping. In respond to this trend, various client-side security mechanisms are developed, such as Content Security Policy (CSP) and HTTP Strict-Transport-Security (HSTS). The presence of these mechanisms on a website can be used as an indicator of the security awareness and practices of that website.

In this section, we report on how the security posture of the European web has evolved over time, by investigating the use of seven security features on European website in September 2013 and September 2015.

8.2.2 Data Collection

To study the security posture of the European web, the popular websites from the 28 member states in the EU are chosen to represent the European web. For each EU country, we selected the top 1,000 websites ending with the corresponding ccTLD (country code top-level domain) from Alexa's list of the top 1 million sites[2]. As a result, we have a set of 23,050 European websites.

The list of ccTLDs of EU countries is shown in Table 8.4.

Table 8.4: ccTLDs of EU countries

ccTLD	Country	ccTLD	Country	ccTLD	Country	ccTLD	Country
at	Austria	ee	Estonia	ie	Ireland	pl	Poland
be	Belgium	es	Spain	it	Italy	pt	Portugal
bg	Bulgaria	fi	Finland	lt	Lithuania	ro	Romania
cy	Cyprus	fr	France	lu	Luxembourg	se	Sweden
cz	Czech Republic	gr	Greece	lv	Latvia	si	Slovenia
de	Germany	hr	Croatia	mt	Malta	sk	Slovakia
dk	Denmark	hu	Hungary	nl	Netherlands	uk	United Kingdom

We then obtained up to 200 webpage URLs for each website by querying the Bing search engine for the popular webpages of each website. After the webpage URLs are obtained, PhantomJS, a headless browser, is used to visit the URLs and retrieve data from webpages. By loading every webpage within PhantomJS, we mimicked the behavior of a regular visitor using a Chrome browser.

After we have crawled all the URLs, we removed the websites with less than 50 successfully crawled pages from our dataset. As a result, our 2013 dataset contains 20,157 websites, and our 2015 dataset describes 18,074, as shown in Table 8.5. The 2015 dataset is smaller than 2013 dataset, which is due to that some domains are disappeared or redirected to other domains.

Table 8.5: Overview of the two datasets (September 2013 and September 2015)

Time	# of websites	# of webpages	avg. # of pages/site
September 2013	20,147	3,499,080	174
September 2015	18,074	2,992,395	166

[2]Alexa top 1 million list in September 2013

8.2.3 Security Features and Scoring System

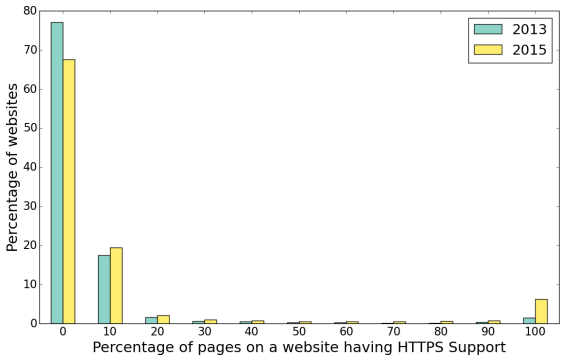
In our study, we focus on the following seven security features which can all be passively detected. We grouped them into three categories:

- 3 features that contribute to **Secure Communication**
- 3 features that mitigate **Cross-Site Scripting**
- 1 feature that enables **Secure Framing**

Category 1: Secure Communication

The Secure Communication category groups three security features: HTTPS support, Secure Cookies, and HTTP Strict Transport Security (HSTS).

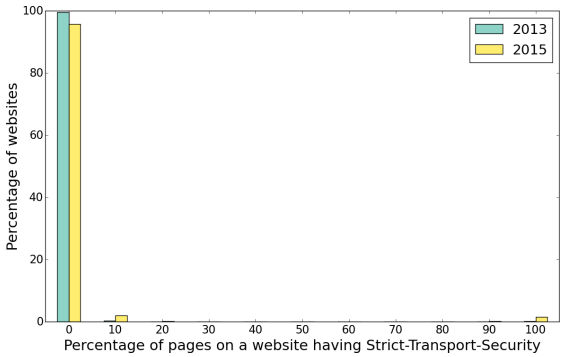
HTTPS Support. The HTTPS protocol is the standard solution for securing web traffic, which guarantees the confidentiality and integrity of web communications by adding the security capabilities of SSL/TLS to HTTP. It also provides website authenticity with the CA/B (Certificate Authority/Browser) trust model.



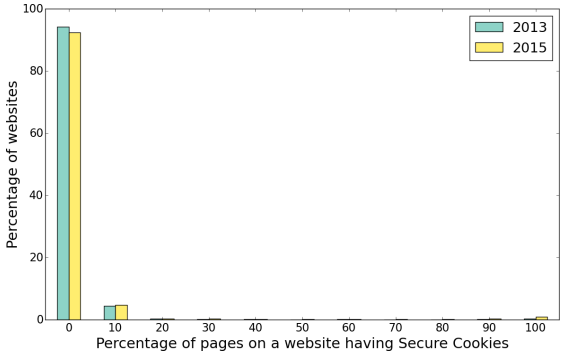
The figure above illustrates how consistently HTTPS is applied on European websites. The X-axis refers to the percentage of pages on a website that apply the technology (grouped together per 10%). The Y-axes depicts the percentage of websites that fall in each of these 10%-groups.

How to read this graph. For instance, in 2013 about 77% of the websites did not use HTTPS at all, where in 2015 this number decreases to 67.5% (illustrated by the two most-left bars). Moreover, only 6.5% of the websites in 2015 are applying HTTPS on all of there pages (compared to 1.5% in 2013), expressed by the two most-right bars in the graph. Finally, just under 20% of the sites are using HTTPS on their domain, but only on up to 10% of their pages (third and fourth bar in the graph).

HTTP Strict-Transport-Security (HSTS). HSTS is designed to mainly prevent SSL-stripping attacks where a secure HTTPS connection is downgraded to a plain HTTP connection by the attacker. Set by a website via a HTTP response header field (`Strict-Transport-Security`), HSTS specifies a period of time during which the user's browser is instructed that all requests to that website need to be sent over HTTPS, regardless of what a user requests. The HSTS Policy helps protecting website users against both passive eavesdropping, as well as active Man-in-the-Middle (MITM) attacks.



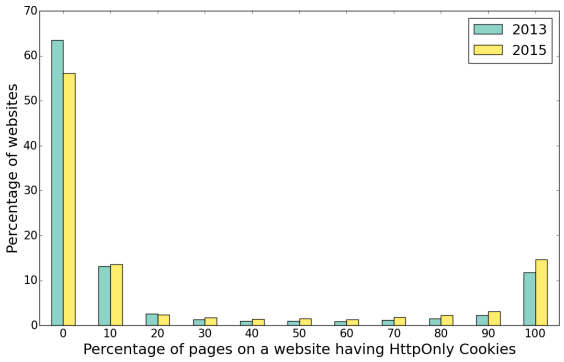
Secure Cookies. HTTPS websites should set the `Secure` attribute when sending cookies to a user's browser, which can prevent cookies from being intercepted by an active network attacker. Although the traffic between a web server and a browser is encrypted when using HTTPS, the cookies stored in the browser are not, by default, limited to an HTTPS context. Thus an active network attacker can intercept any outbound HTTP request from the browser and redirect that request to the same website over HTTP in order to reveal the cookies [32]. By setting the `Secure` attribute, the scope of a cookie is limited to secure channels, thus stopping browsers from sending cookies over unencrypted HTTP requests.



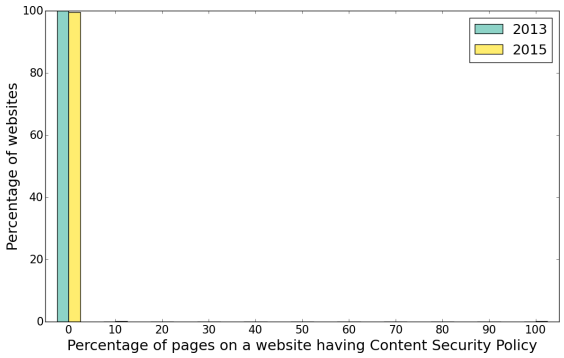
Category 2: XSS Mitigation

The XSS Mitigation category groups three security features: HTTPOnly Cookies, the Content Security Policy (CSP) and X-Content-Type-Options (XCTO).

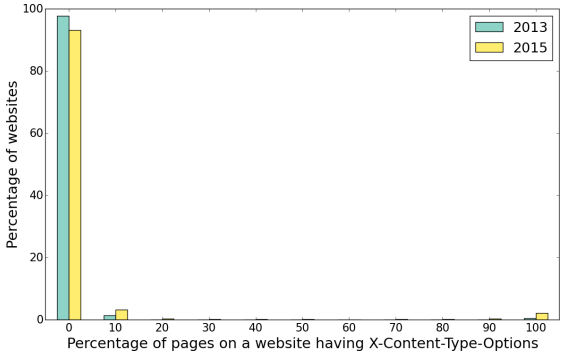
HttpOnly Cookies. First introduced in Internet Explorer (IE) 6 SP1, the `HttpOnly` attribute is designed to mitigate the risk of malicious client-side scripts accessing sensitive cookie values. Cookies are accessible to JavaScript code by default, which allows attackers to steal the cookies via an XSS attack. Using the `HttpOnly` attribute in a `Set-Cookie` header restrict the access of that cookie to the HTTP(S) protocol, making it inaccessible to client-side JavaScript [32].



Content Security Policy (CSP). CSP provides a standard HTTP response header (`Content-Security-Policy`) that allows a webpage to declare approved sources of content that browsers should be allowed to load on that specific page. Whenever a requested resource originates from a source that is not defined in the CSP, it will simply not be loaded [188]. For example, if the policy does not allow in-line JavaScript, then, even if an attacker is able to inject malicious JavaScript in the webpage, the injected code will not be executed.



X-Content-Type-Options (XCTO). Internet Explorer has a MIME-sniffing feature that will attempt to determine the content type for each downloaded resource. This feature, however, can lead to security problems for servers hosting untrusted content. To prevent Internet Explorer from MIME-sniffing, thus reducing exposure to attacks, a web server can send the `X-Content-Type-Options` response header with the `nosniff` value.



Category 3: Secure Framing

The Secure Framing category reports on the use of X-Frame-Options (XFO).

X-Frame-Options (XFO). The HTTP response header `X-Frame-Options` is designed to mitigate Clickjacking attacks [176]. In a Clickjacking attack, the attacker redresses the user interface of website A with transparent layers, and then trick the user into clicking on a button on an embed page from website B when they were intending to click on the the same place of the overlaying page from website A. To stop Clickjacking attacks, the `X-Frame-Options` header can be used to instruct a user's browser whether a certain page is allowed to be embedded in a frame. For example, if the `X-Frame-Options` header's value is `DENY`, then the browser will prevent the page from rendering when embedded within a frame.

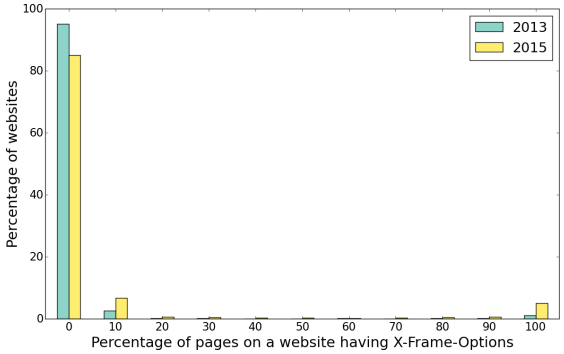


Figure 8.4: The use of XFO on European web in 2013 and 2015

Overview on the use of security features on European web in 2013 and 2015

Table 8.6 gives an overview of the use of security features on European web in 2013 and 2015. It clearly illustrates that the web security on the European Web did improve, as each of the security features have been adopted in 2015 by a larger fraction of websites than in 2013.

Table 8.6: Overview of the use of security features on European web

Security feature	**% of websites**		**Improvement**
	Sept. 2013	Sept. 2015	
HTTPS Support	22.96%	32.40%	9.44%
Secure Cookies	5.86%	7.56%	1.70%
HTTP Strict-Transport-Security	0.49%	4.30%	3.81%
HttpOnly Cookies	36.52%	43.86%	7.34%
X-Content-Type-Options	2.24%	6.82%	4.58%
Content Security Policy	0.05%	0.43%	0.38%
X-Frame-Options	4.80%	14.93%	10.13%

8.2.4 Web Security Scoring System

In order to compare the security level among different websites, and among different countries or business verticals (represented by the websites of each country or business vertical), we developed a web security scoring system that gives a quantitative security score for each website.

For each (group of) website(s), we define the overall security score ($OverallScore$) as a weighted average of three distinct subscores:

$$\begin{aligned}\textbf{OverallScore} = &\frac{40}{100} \times SecureCommunicationScore \\ &+ \frac{40}{100} \times XSSMitigationScore \\ &+ \frac{20}{100} \times SecureFramingScore\end{aligned}$$

Calculation of subscores

As part of of scoring system, we assess for each security feature how well the (group of) website(s) is doing compared to websites in the full dataset. For instance, we want to grade a website with a score 0.61 to the feature HTTPS, if the website outperforms 61% of the websites in our dataset (i.e. by having a higher percentage of pages over HTTPS). The scores of the individual features are then combined to provide a metric for the three subscores.

More concretely, we apply the following approach:

1. For each security feature, we compute an empirical cumulative function (ECDF) for all websites. The ECDF is computed based on the percentage of webpages having that feature on a particular website.
2. This computed ECDF is used to calculate an ECDF value per website and per feature.
3. The subscores are calculated by applying a weighted averages of the ECDF values.

In particular, the following weights are used to calculated the three subscores:

Secure Communication Score. This subscore is measured by applying a weighted average of the HTTPS, HSTS, and Secure Cookies usage.

$$SecureCommunicationScore = \frac{45}{100} \times HTTPS + \frac{25}{100} \times SecureCookies + \frac{30}{100} \times HSTS$$

XSS Mitigation Score. This subscore measured by applying a weighted average of the HttpOnly Cookies, XCTO, and CSP usage.

$$XSSMitigationScore = \frac{50}{100} \times HttpOnlyCookies + \frac{25}{100} \times XCTO + \frac{25}{100} \times CSP$$

Secure Framing Score. This subscore is measured by the XFO usage.

$$SecureFraming = \frac{100}{100} \times XFO$$

8.2.5 EU Web Security Score, in terms of website popularity

To assess the web security score in terms of website popularity, the websites are grouped per 10,000 Alexa ranks, and the average score is calculated for websites that belongs to that rank range. Figure 8.5 shows the average *OverallScore* for per 10k Alexa ranks.

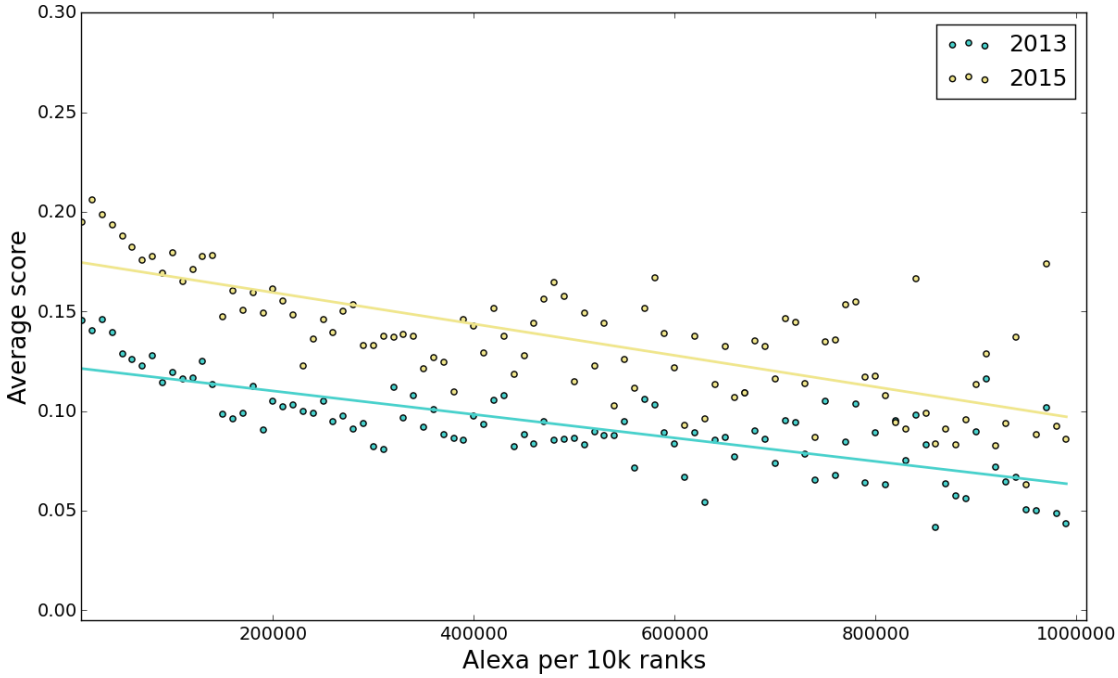


Figure 8.5: The average *OverallScore* for per 10k Alexa ranks

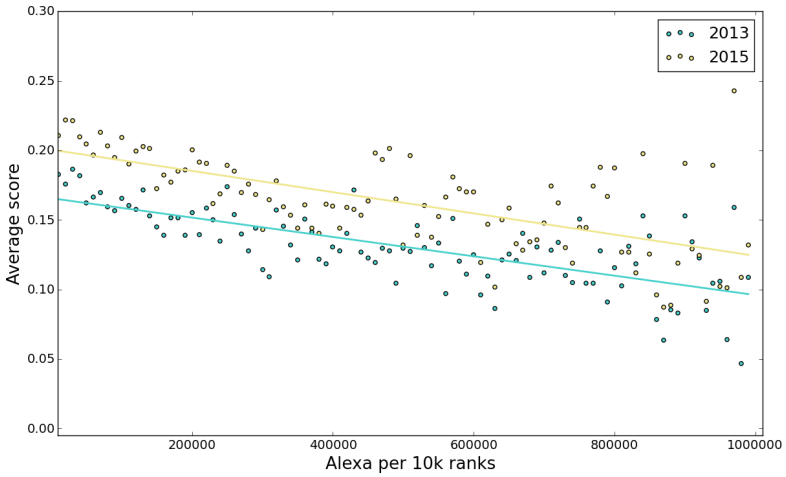
Figure 8.5 hints that higher ranked websites tend to have higher score. To confirm this assumption, the Spearman correlation is used to assert the correlation between the *OverallScore* in a website and its Alexa rank for the 2013 dataset and 2015 dataset (as listed in Table 8.7).

Spearman's rank correlation coefficient is a nonparametric measure of the monotonicity of the relationship between two variables. It is defined as the Pearson correlation coefficient between the ranked variables. However, unlike the Pearson correlation, the Spearman correlation does not assume that both variables are normally distributed. It is a nonparametric statistic, which do not rely on assumptions that the dataset is drawn from a given probability distribution. The result of Spearman correlation varies between −1 and +1, and a positive coefficient implies that as one variable increases, the other variable also increases and vice versa. When using Spearman correlation to test statistical dependence, we set the significance level to 5%. The p-value is calculated using Student's t-distribution. We accept the hypothesis only if the p-value is smaller than the significance level.

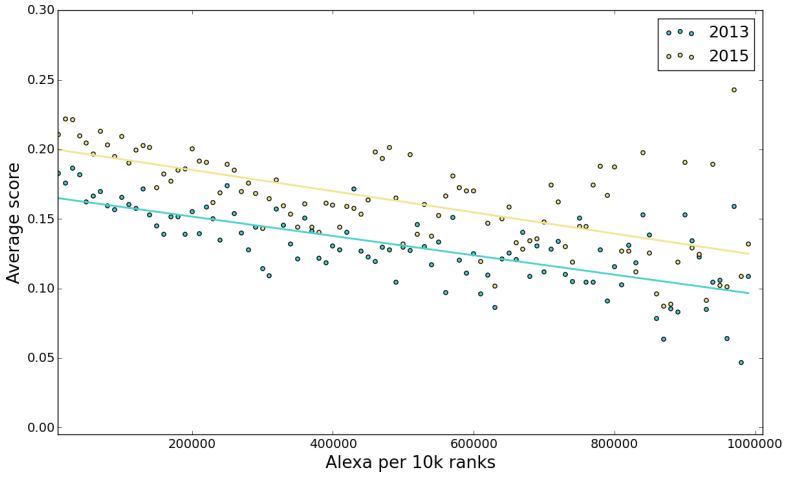
As expected from Figure 8.5, there is negative correlation between the *OverallScore* in a website and its Alexa rank (see Table 8.7), and this correlation also holds for all three sub scores.

Table 8.7: Correlation between the *OverallScore* in a website and its Alexa rank

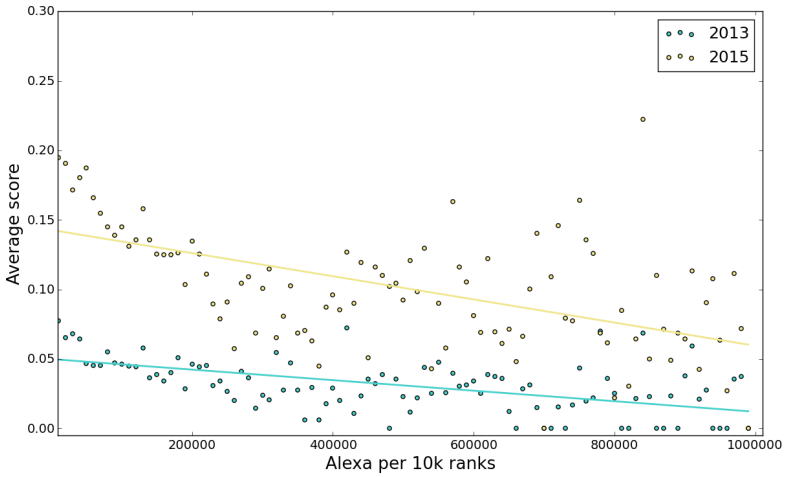
Correlation 2013 Dataset		Correlation 2015 Dataset	
coefficient	p-value	coefficient	p-value
−0.11	1.81×10^{-59}	−0.08	3.84×10^{-31}



(a) SecureCommunicationScore



(b) XSSMitigationScore



(c) SecureFramingScore

Figure 8.6: Average subscores per 10k Alexa ranks

Websites that adopting more security features

In the previous section , we illustrated how a website evolves by calculating ECDF-based scores, which essentially compare the usage of a security feature on a particular website with the usage on other websites in the dataset. In this section, we assess to what extent the security of a particular website did improve over time, by measuring for each website if it adopts more security features over time or not (and thus not how consistently a security features is applied on a given domain). By doing this, we found 6,512 websites that adopted more security features in 2015 than what they already have in 2013. Since higher ranked websites get higher scores (as shown in the previous section), we expect that higher ranked websites tend to adopt more security features. To confirm this hypothesis, we use Point-biserial correlation to study the correlation between the adoption of security features in a website and its Alexa rank.

Generally, Pearson product-moment correlation coefficient (Pearson's r) is widely used in statistics as a measure of the degree of linear dependence between two quantitative variables. In our case, the adoption of security feature is a binary choice, thus we use the Point-biserial correlation coefficient. The Point-biserial correlation coefficient is a special case of Pearson in which one variable is quantitative and the other variable is dichotomous. The result of Point-biserial correlation varies between −1 and +1, and a positive coefficient implies that as one variable increases, the other variable also increases and vice versa. When using Point-biserial correlation to test statistical dependence, we set the significance level to 5%. The p-value is calculated using Student's t-distribution. We accept the hypothesis only if the p-value is smaller than the significance level.

The resulting Point-biserial correlation coefficient is -0.08, with p-value 6.24×10^{-29}, which confirms our hypothesis that higher ranked websites tend to adopt more security features. To better illustrate this correlation, for per 1,000 Alexa ranks, we calculate the percentage of websites that belongs to that rank range, which have adopted more security features, as shown in Figure 8.7.

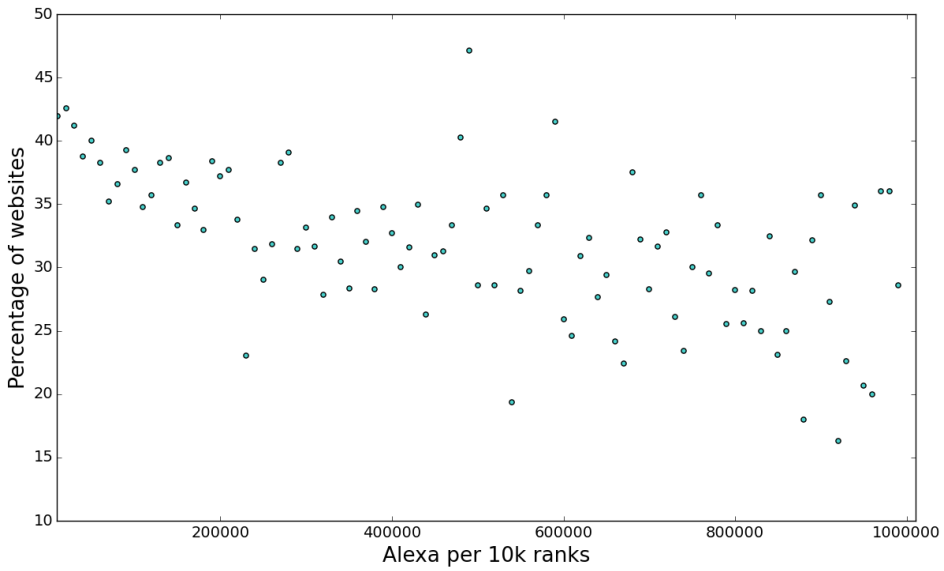


Figure 8.7: The percentage of websites that adopted more security features in 2015 versus 2013, plotted per 10k Alexa ranks

8.2.6 Web Security Score per country in EU

In this section, we compare the security evolution of the websites per country. For each EU country, the average score is calculated for websites that belongs to that country. Figure 8.8 shows the average *OverallScore* for 25 EU countries. Cyprus(.cy), Malta(.mt) and Luxemburg(.lu) were removed from the dataset, as the number of websites in these countries were less than 100. The countries in Figure 8.8 are sorted by their 2013 security score, to easily identify countries that got better than their adjacent peers. Similarly, Figure 8.10 displays the individual subscores.

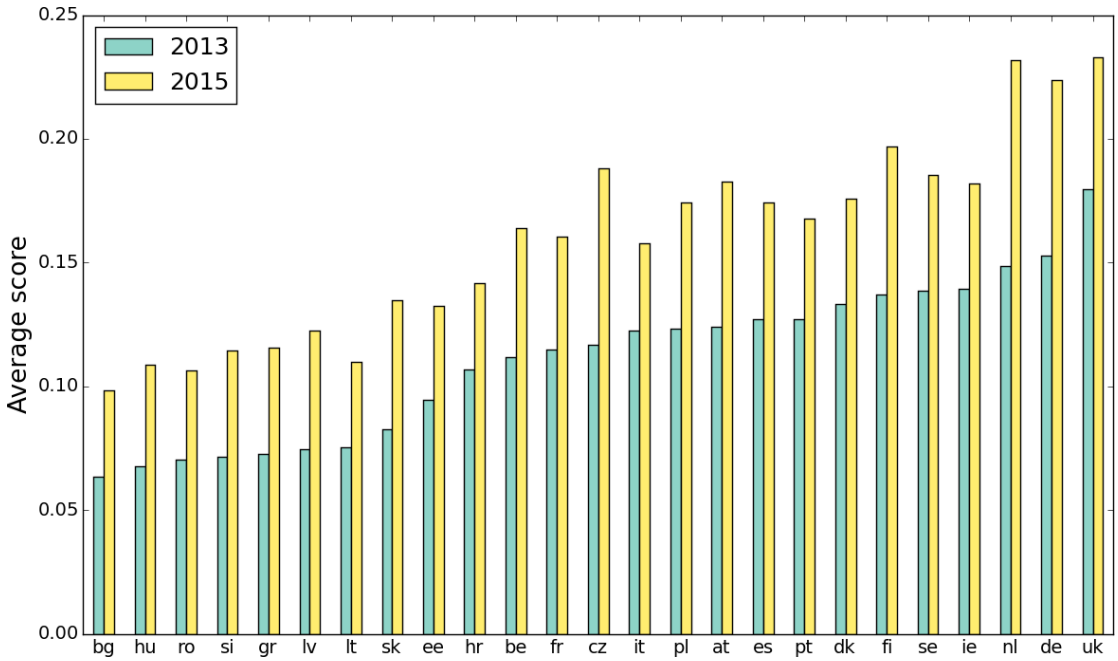


Figure 8.8: The average *OverallScore* for each country

In Figure 8.9, we assess to what extent the security of a particular website did improve over time, by measuring for each website if it adopts more security features over time or not. Figure 8.9 expresses the percentage of websites per country that adopted more security features in 2015, than in 2013. The countries are sorted, based on their OverallScore of 2013.

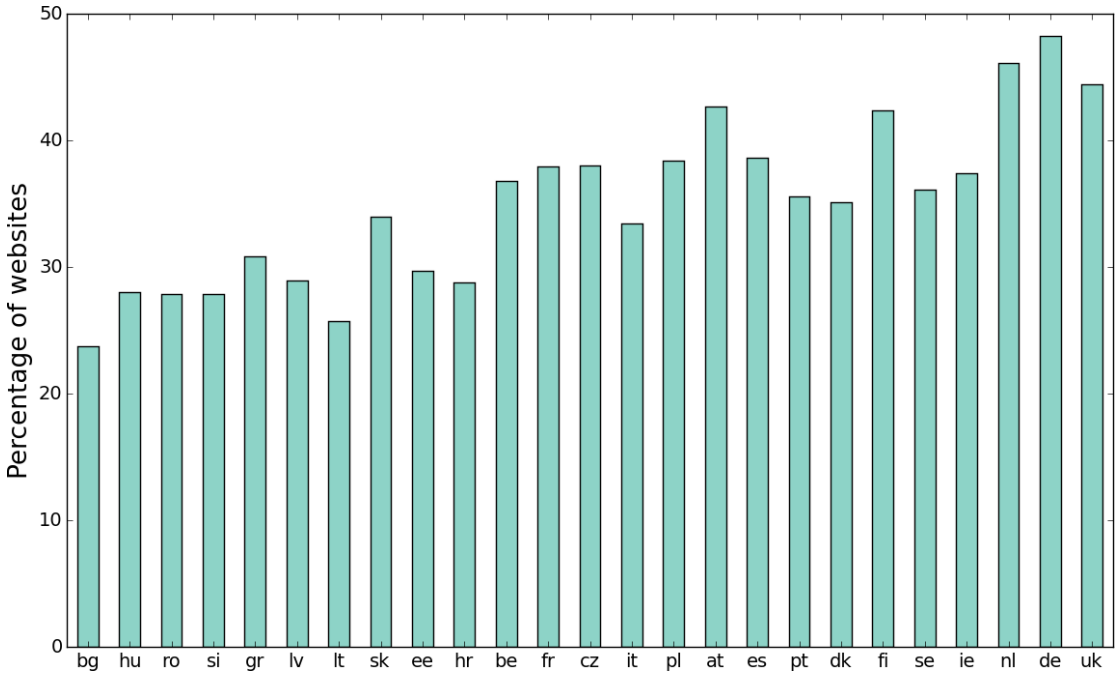


Figure 8.9: The percentage of websites that adopted more security features in 2015 versus 2013, grouped per country

The top 10 countries in terms of OverallScore 2013 and 2015 are listed below. The Czech Republic and Poland (printed in italic) entered the top 10 in 2015, at the cost of Spain and Portugal.

Germany, The Netherlands and the United Kingdom deserve an honourable mentioning. Not only do they represent the top 3 in both 2013 and 2015, these three countries also are top 5 in each of the subscores (both in 2013 as well as 2015)!

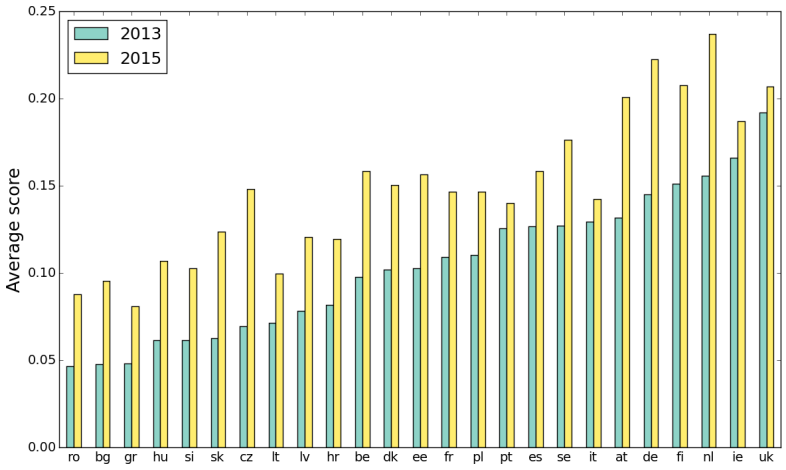
Overall ranking 2013

1. United Kingdom
2. Germany
3. The Netherlands
4. Ireland
5. Sweden
6. Finland
7. Danmark
8. *Portugal*
9. *Spain*
10. Austria

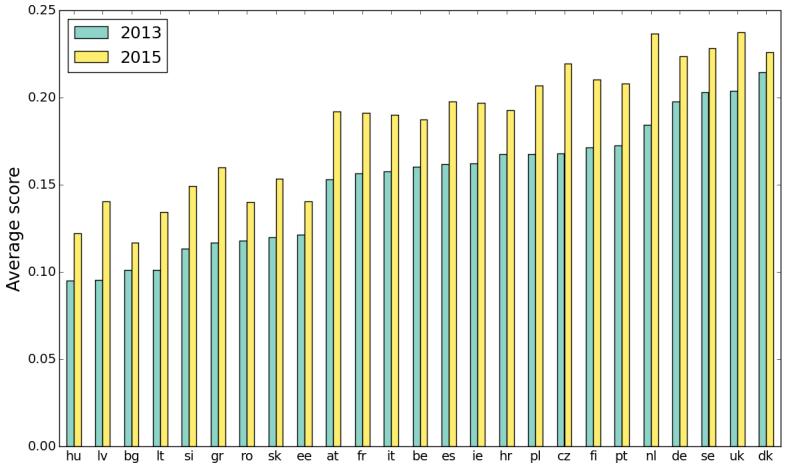
Overall ranking 2015

1. United Kingdom
2. The Netherlands
3. Germany
4. Finland
5. *Czech Republic*
6. Sweden
7. Austria
8. Ireland
9. Danmark
10. *Poland*

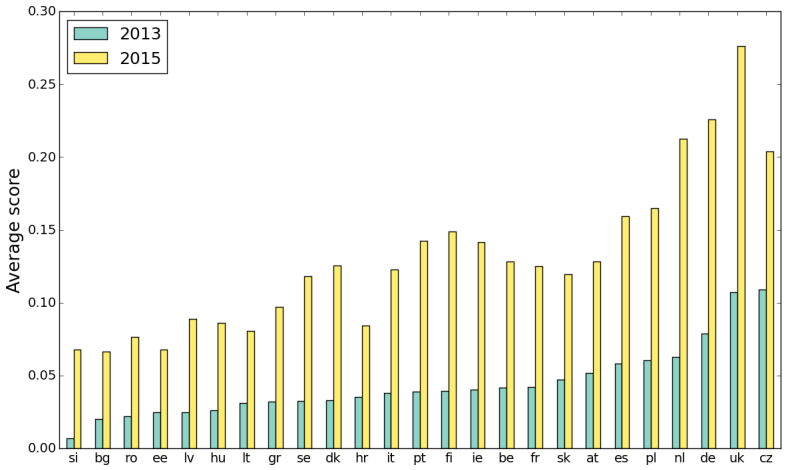
A more detailed report per country can be found in Figure 8.2, which expresses the ECDF values of each security features in 2013 and 2015.



(a) SecureCommunicationScore

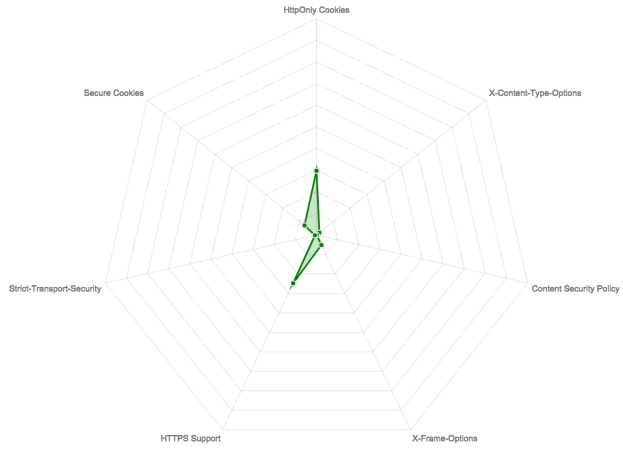


(b) XSSMitigationScore

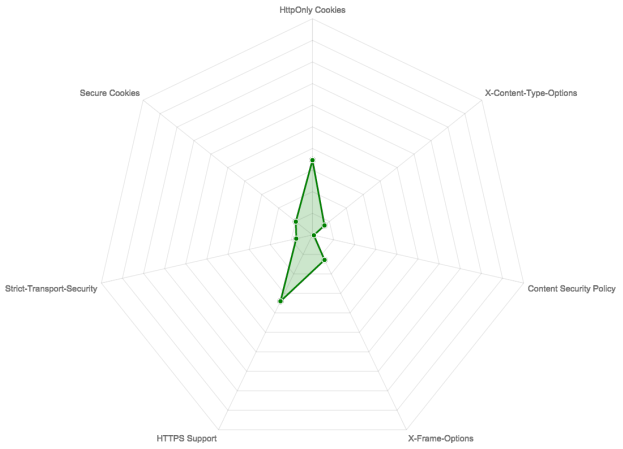


(c) SecureFramingScore

Figure 8.10: Average subscores per country

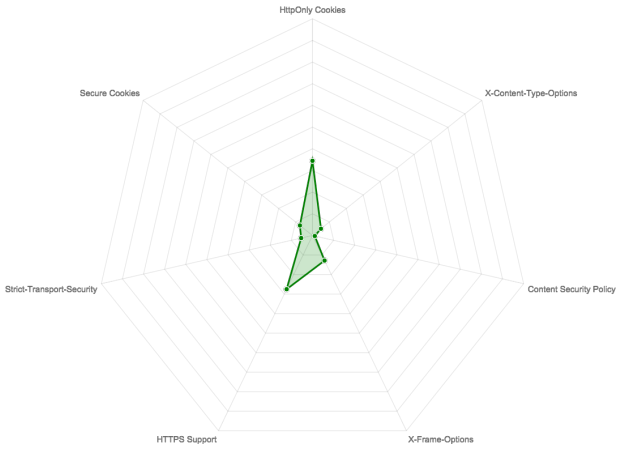


(a) Austria (2013)



(b) Austria (2015)

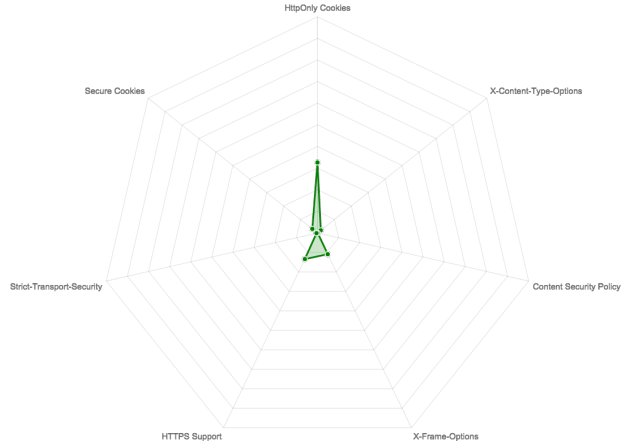
(c) Belgium (2013)



(d) Belgium (2015)

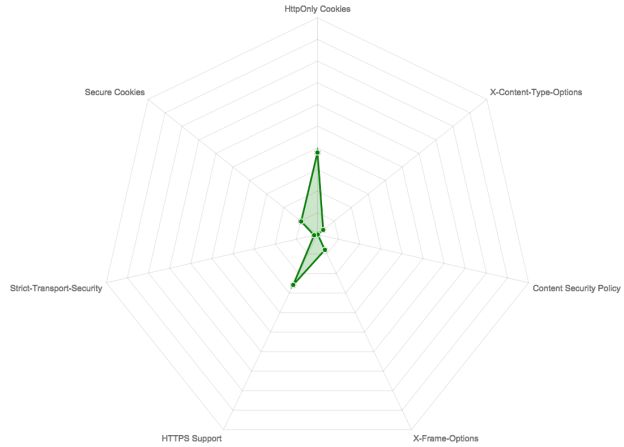
(e) Bulgaria (2013)

(f) Bulgaria (2015)

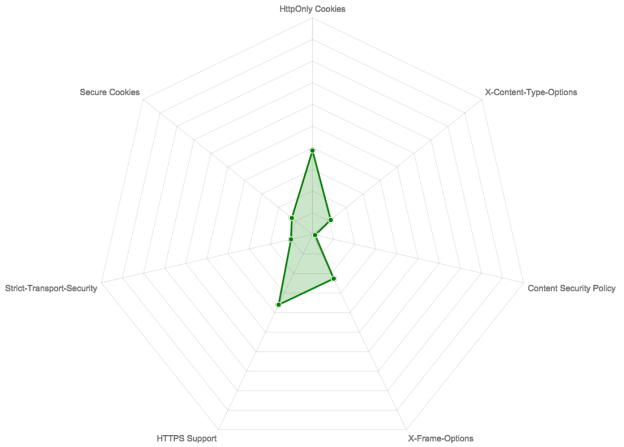


(g) Czech Republic (2013)

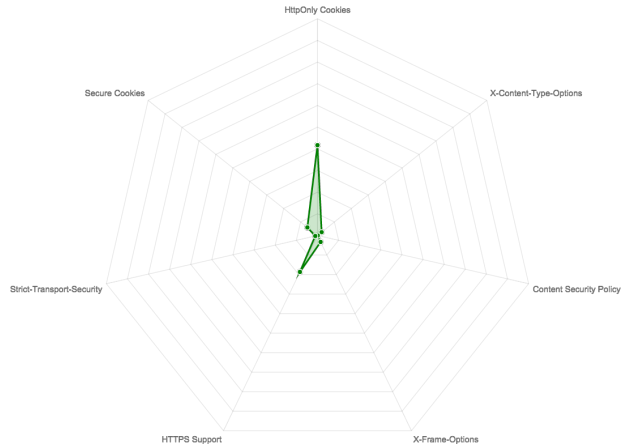
(h) Czech Republic (2015)



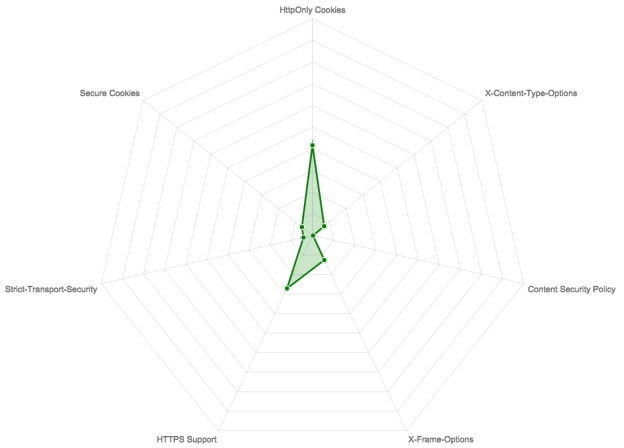
(i) Germany (2013)



(j) Germany (2015)



(k) Denmark (2013)

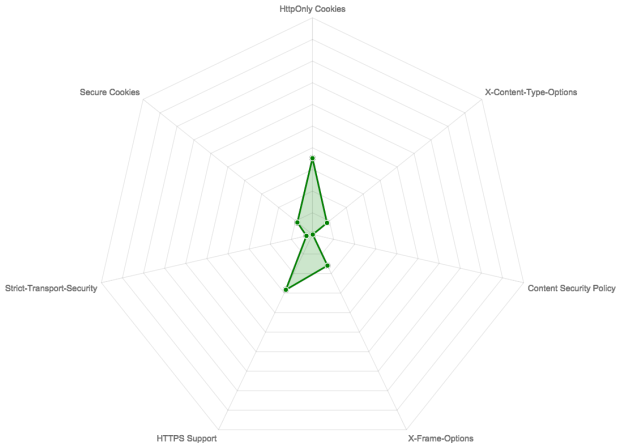


(l) Denmark (2015)

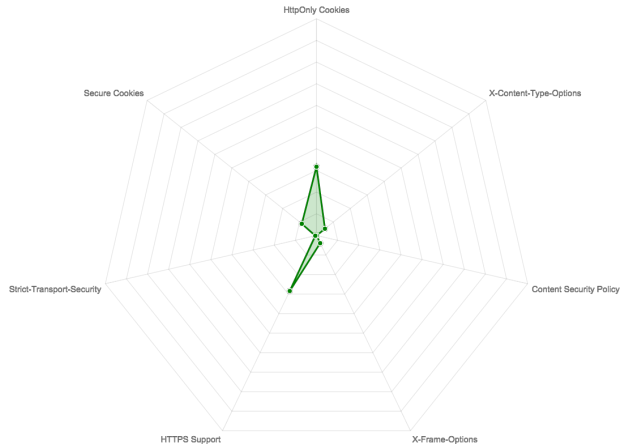
(m) Estonia (2013)

(n) Estonia (2015)

(o) Spain (2013)

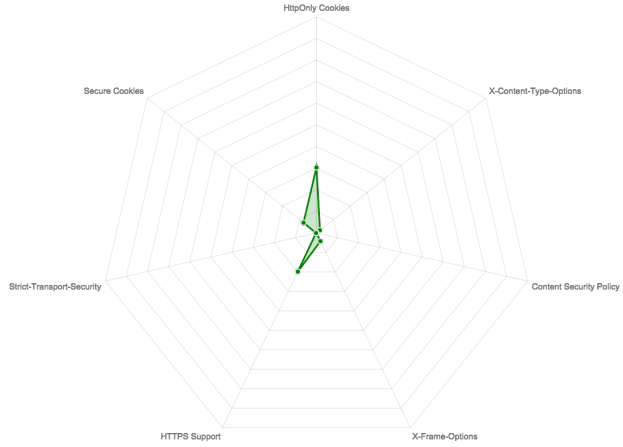


(p) Spain (2015)



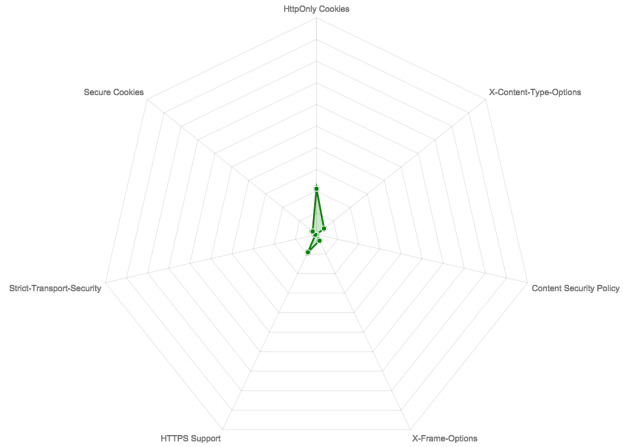
(q) Finland (2013)

(r) Finland (2015)



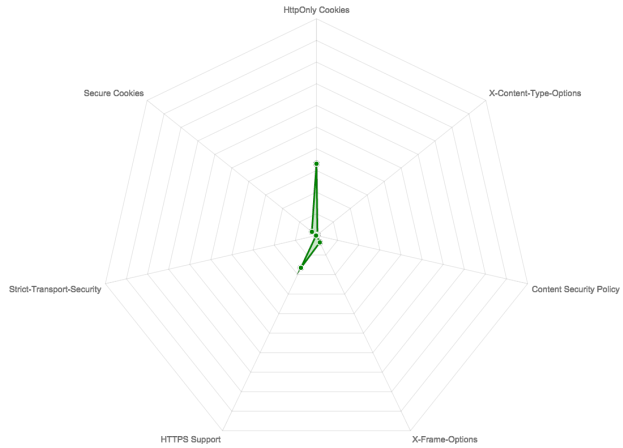
(s) France (2013)

(t) France (2015)



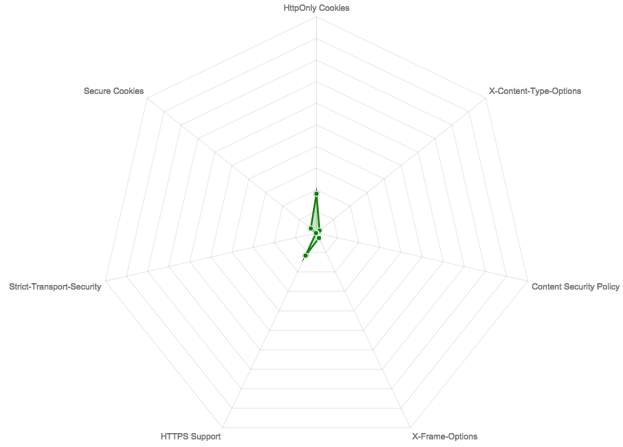
(u) Greece (2013)

(v) Greece (2015)



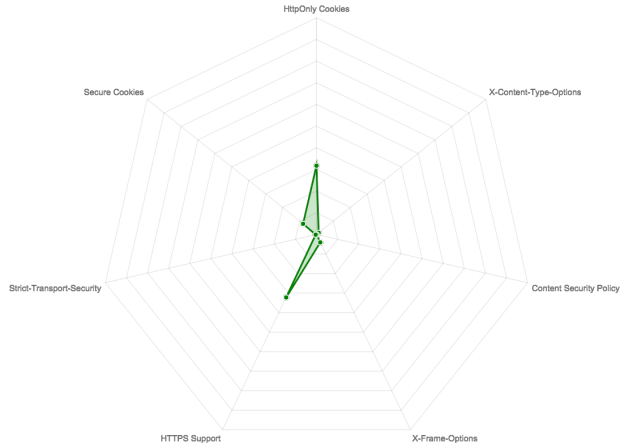
(w) Croatia (2013)

(x) Croatia (2015)



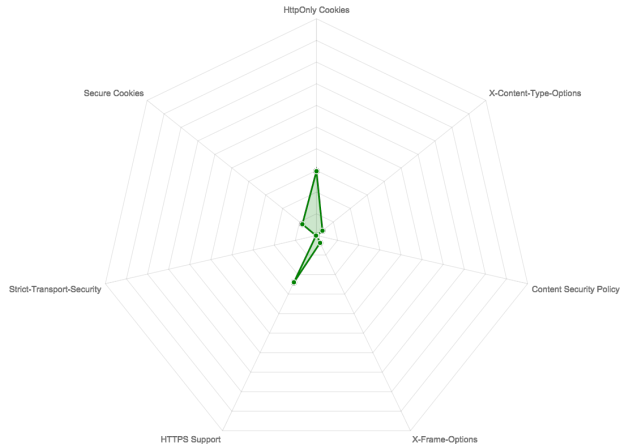
(y) Hungary (2013)

(z) Hungary (2015)



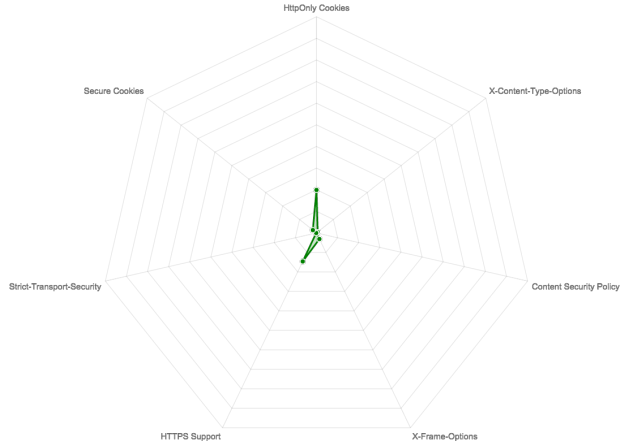
(aa) Ireland (2013)

(ab) Ireland (2015)



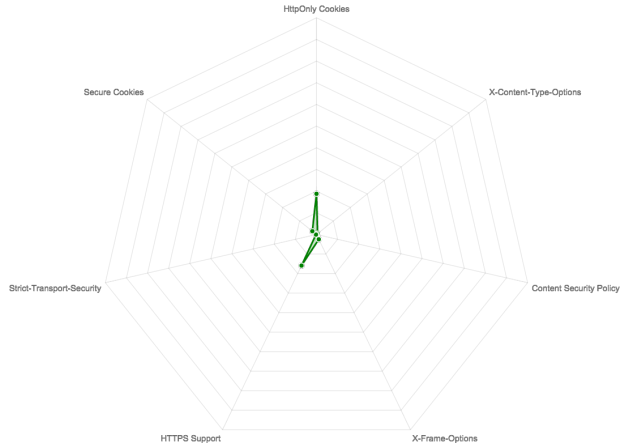
(ac) Italy (2013)

(ad) Italy (2015)



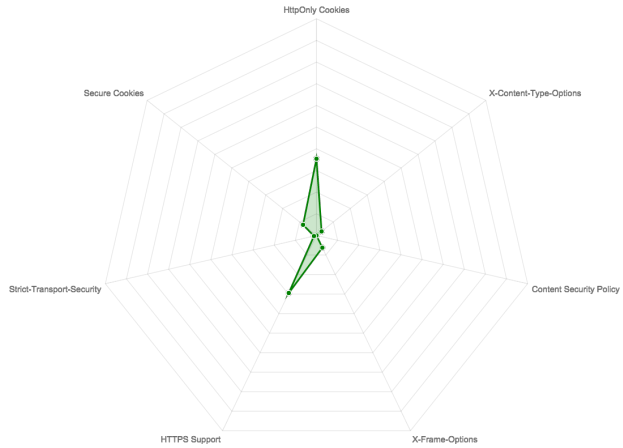
(ae) Lithuania (2013)

(af) Lithuania (2015)

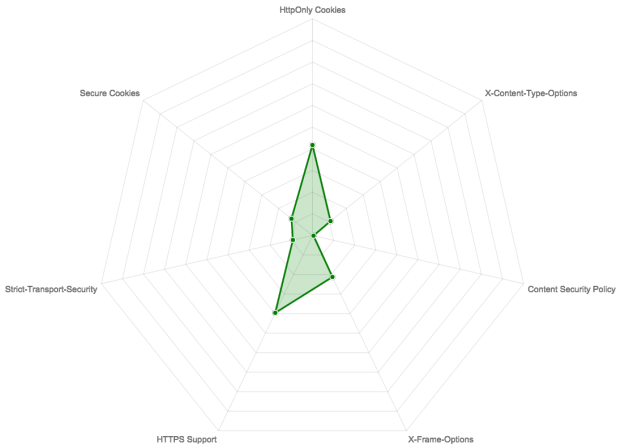


(ag) Latvia (2013)

(ah) Latvia (2015)

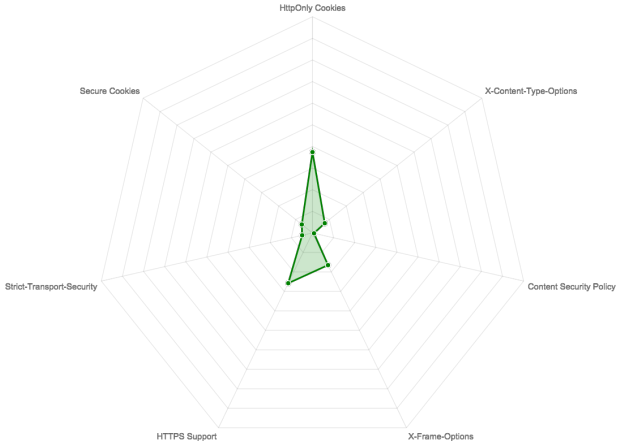


(ai) The Netherlands (2013)



(aj) The Netherlands (2015)

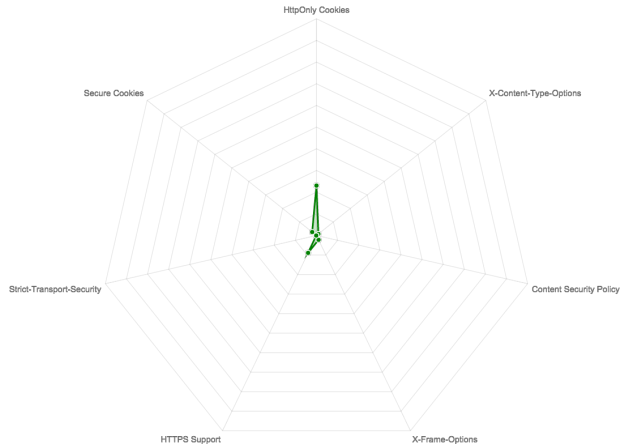
(ak) Poland (2013)



(al) Poland (2015)

(am) Portugal (2013)

(an) Portugal (2015)

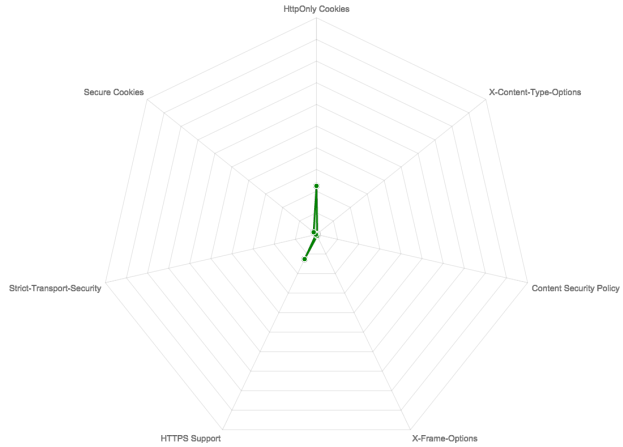


(ao) Romania (2013)

(ap) Romania (2015)

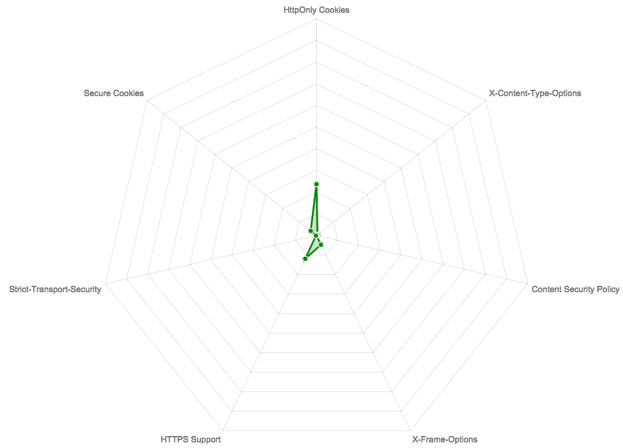
(aq) Sweden (2013)

(ar) Sweden (2015)



(as) Slovenia (2013)

(at) Slovenia (2015)



(au) Slovakia (2013)

(av) Slovakia (2015)

(aw) United Kingdom (2013)

(ax) United Kingdom (2015)

Figure 8.2: Per country comparison between 2013 and 2015

8.2.7 Web Security Score per business vertical in EU

In this section, we compare the security evolution of the websites per business vertical[3]. For the ten most popular business vertical, the average score is calculated for websites that belongs to that business vertical.

Figure 8.3 shows the average *OverallScore* for 10 business verticals, sorted by their 2013 security score, to easily identify business verticals that got better than their adjacent peers. Similarly, Figure 8.5 displays the individual subscores.

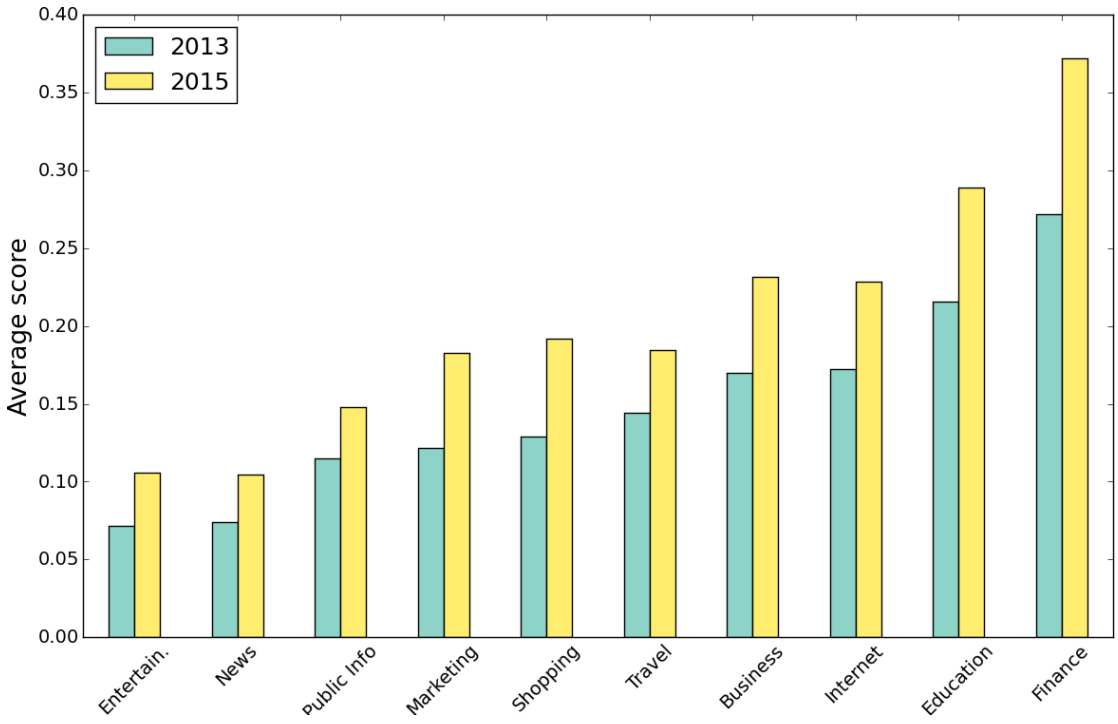


Figure 8.3: The average *OverallScore* for each business vertical

In Figure 8.4, we assess to what extent the security of a particular website did improve over time, by measuring for each website if it adopts more security features over time or not. Figure 8.4 expresses the percentage of websites per business vertical that adopted more security features in 2015, than in 2013. The business verticals are sorted, based on their OverallScore of 2013.

[3]the business vertical of each websites is derived from McAfee's TrustedSource Web Database [5]

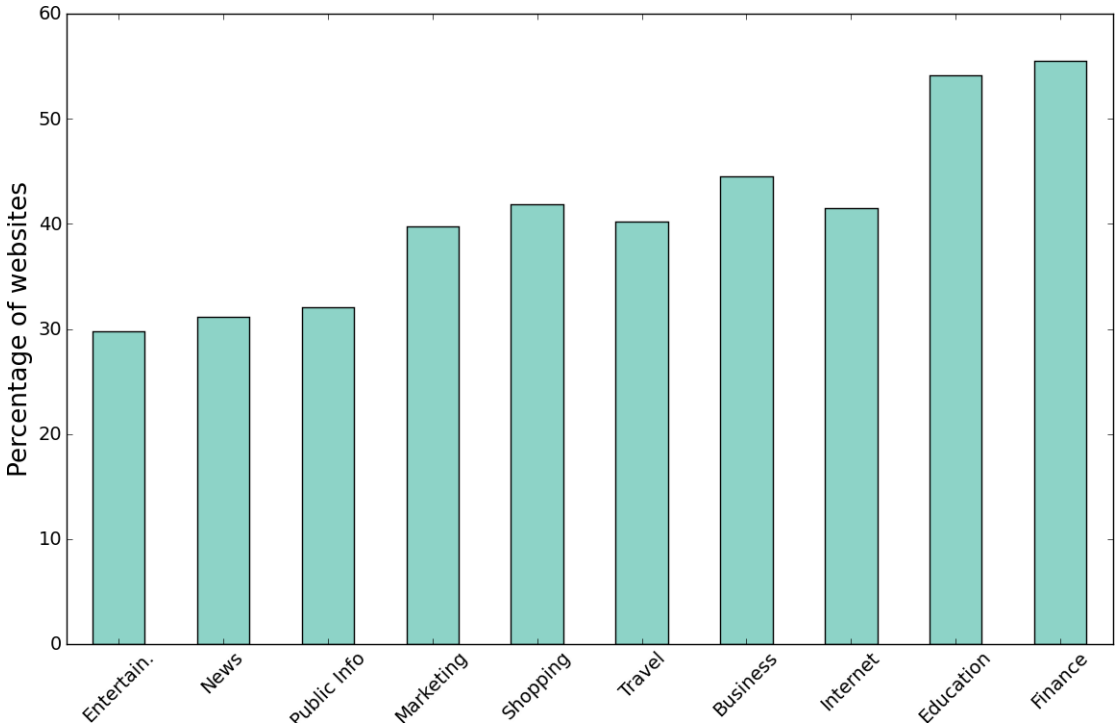
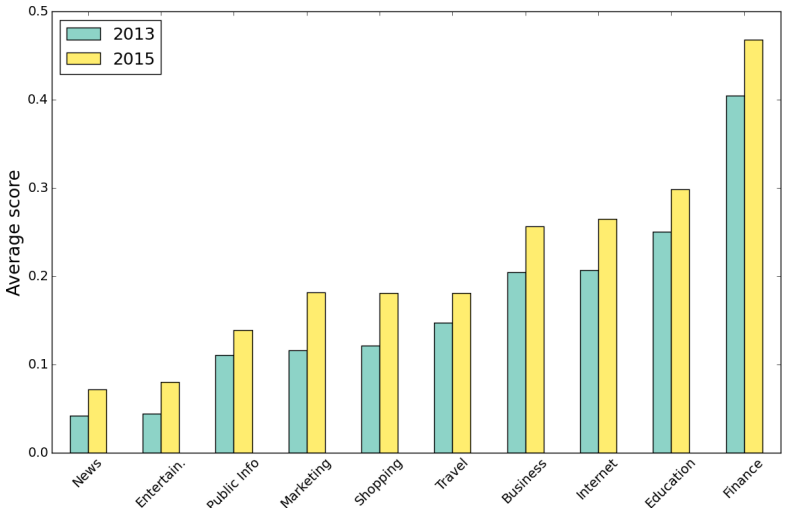


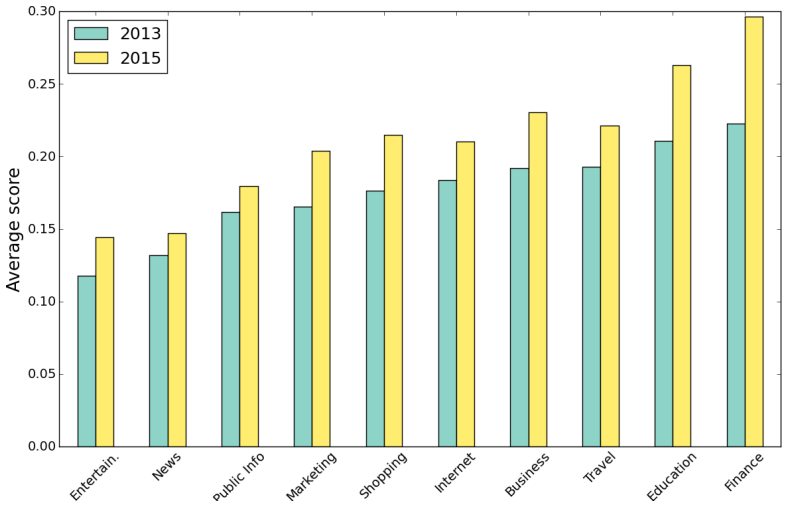
Figure 8.4: The percentage of websites that adopted more security features in 2015 versus 2013, grouped per business vertical

The *Education* and *Finance* verticals are the two best performing categories in each of the subscores and the Overall score. Moreover, notice that in Figure 8.3 and 8.5 the *Finance* vertical was already outperforming the other categories in 2013, apart from secure framing (via X-Frame-Options). In the dataset of 2015, the *Finance* vertical did catch up with the *Education* vertical in secure framing, and is now leading each of the subscores as well as the OverallScore.

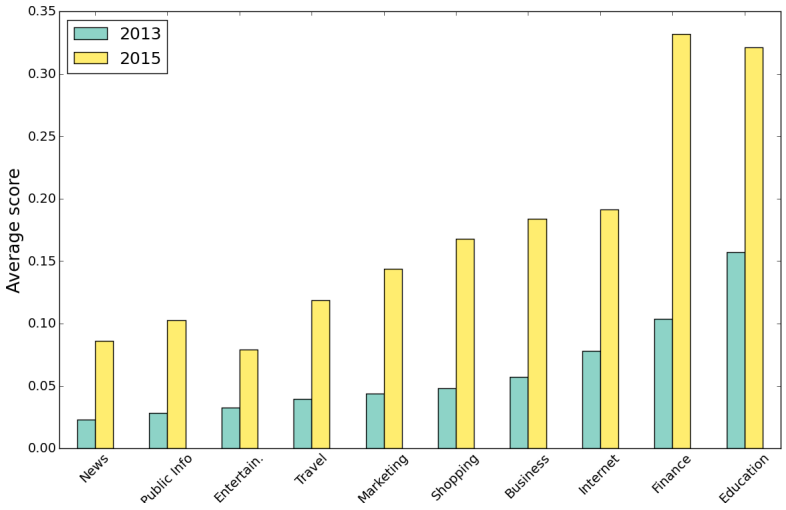
A more detailed report per business vertical can be found in Figure 8.2, which expresses the ECDF values of each security features in 2013 and 2015.



(a) SecureCommunicationScore

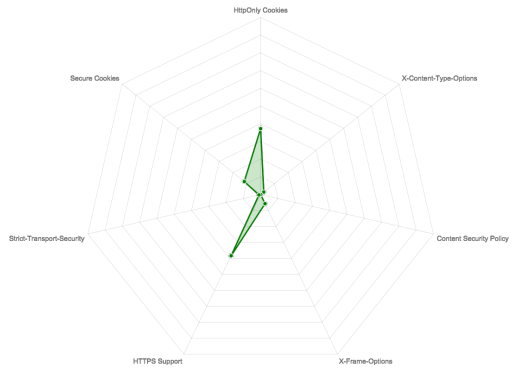


(b) XSSMitigationScore



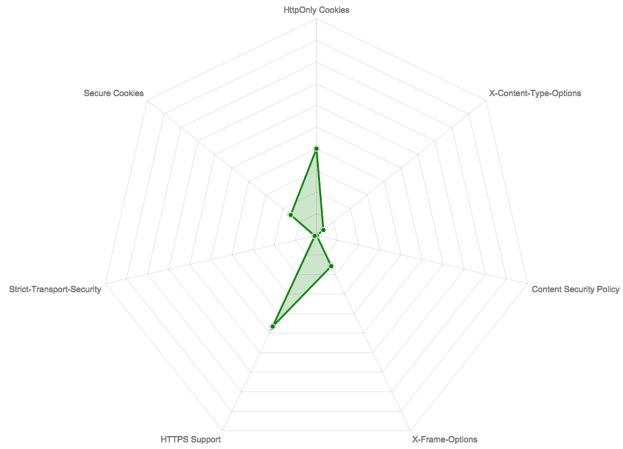
(c) SecureFramingScore

Figure 8.5: Average subscores per business vertical

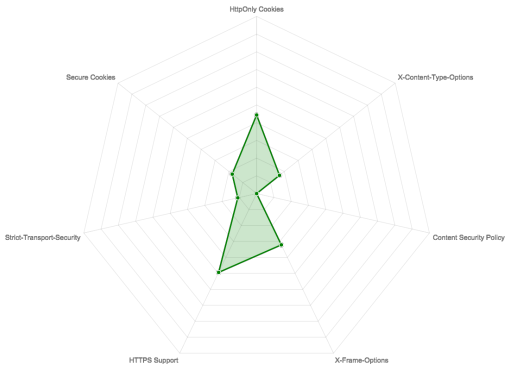


(a) Business (2013)

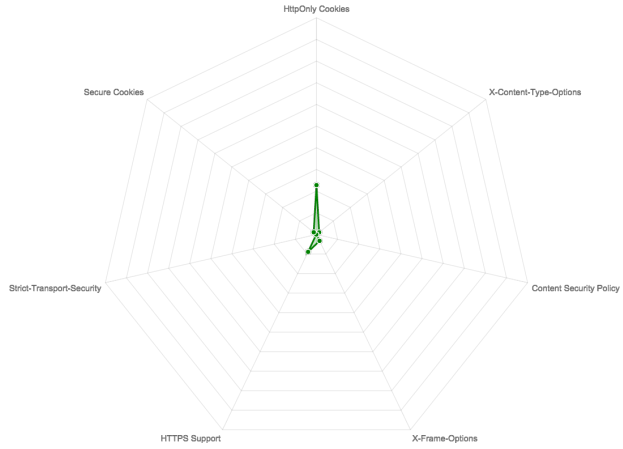
(b) Business (2015)



(c) Education (2013)

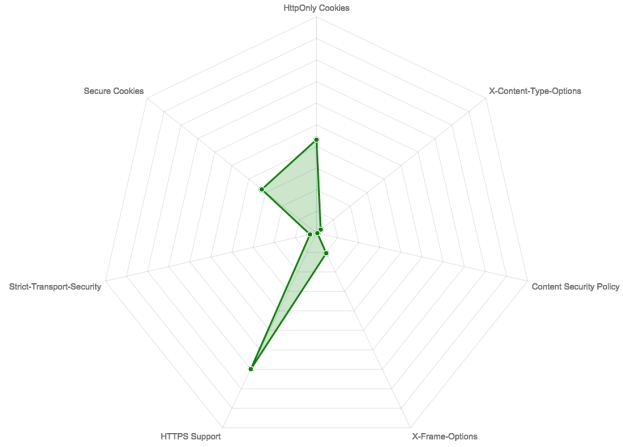


(d) Education (2015)

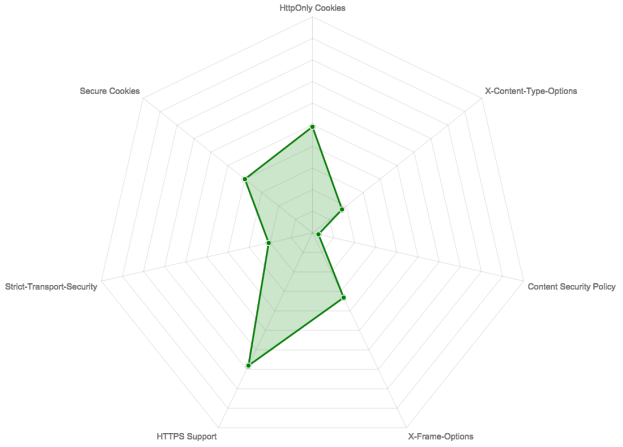


(e) Entertainment (2013)

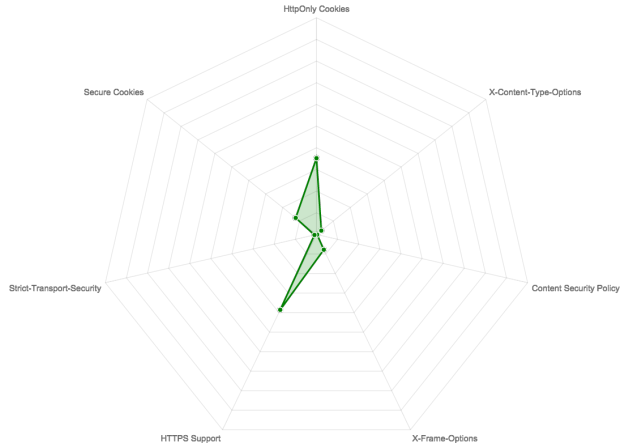
(f) Entertainment (2015)



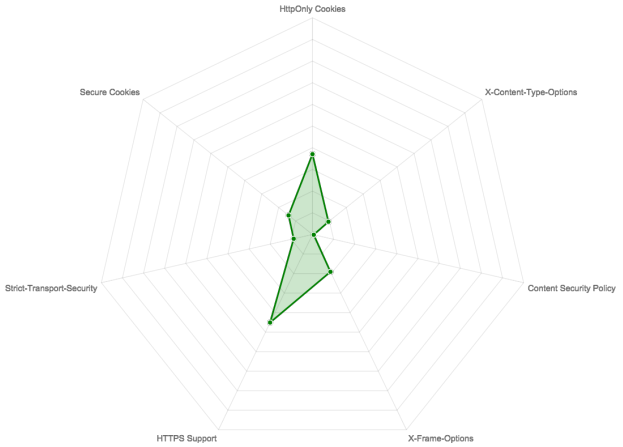
(g) Finance (2013)



(h) Finance (2015)

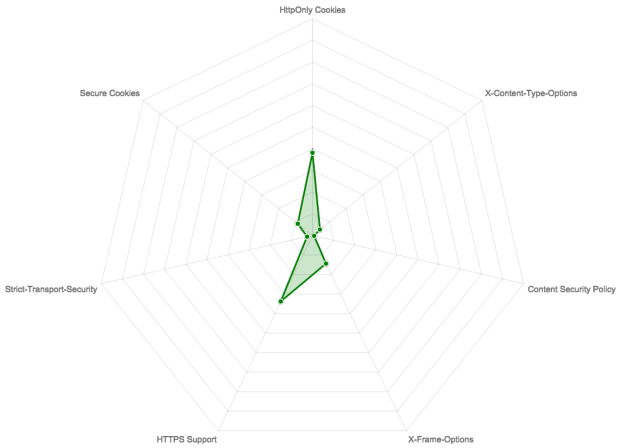


(i) Internet (2013)

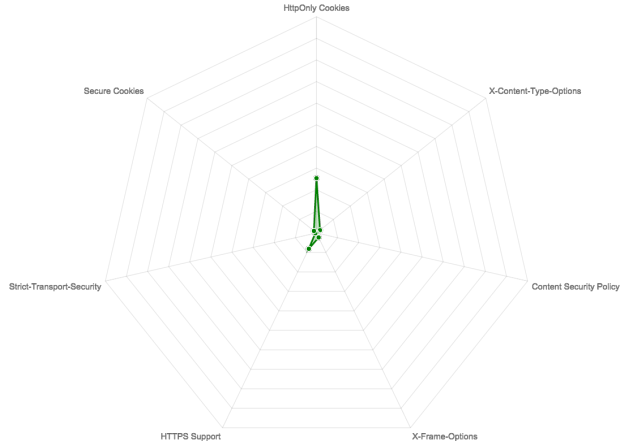


(j) Internet (2015)

(k) Marketing (2013)



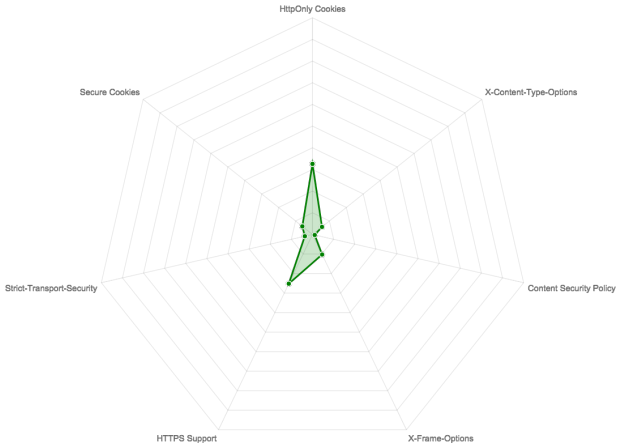
(l) Marketing (2015)



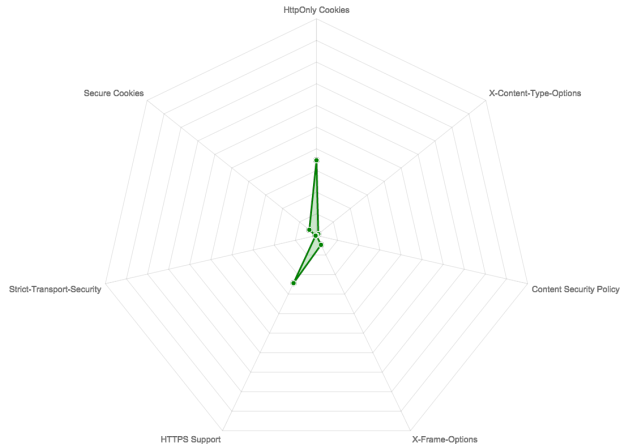
(m) News (2013)

(n) News (2015)

(o) Public info (2013)



(p) Public info (2015)



(q) Shopping (2013)

(r) Shopping (2015)

(s) Travel (2013)

(t) Travel (2015)

Figure 8.2: Per business vertical comparison between 2013 and 2015

8.2.8 Conclusions

In this section, we have compared the security state of practise in the European Web, both in September 2013 and September 2015. To do so, we crawled about 3 million web pages of 18,000 websites. Based on analysis of available data of the crawling experiment, we could observe the following longitudinal trends:

First, we have observed that the most popular websites (according to the Alexa ranking) have a higher web security metric that less popular websites. Moreover, we could validate that the most popular websites were adopting new security features quicker that less popular websites in the two year timeframe.

Second, the United Kingdom, the Netherlands and Germany are the three best performing countries in EU with respect to the use of web security features, both in 2013 as well as in 2015. Moreover, the Czech Republic and Poland entered the top 10 in 2015, at the cost of Spain and Portugal. In addition, we identified a trend that websites in countries scoring better in 2013, also tend to adopt more new security features in 2015.

Third, by examining the websites based on their business vertical, we can state that the websites in the Finance and Education category are outperforming other verticals in the data set, with respect to the web security metric. In addition, we identified a trend that websites in business verticals scoring better in 2013, also tend to adopt more new security features in 2015.

8.3 Large-scale analysis on client-side complexity based on DOM-based XSS measurements

8.3.1 Motivation

The recent years have brought a significant push of application logic from the server to the client-side. Modern Web applications consist of a considerable amount of JavaScript code that is executed in the user's browser. Only little inside exists on the complexity of this code and the security implications, that are connected with this shift.

Hence, to gain insight into this topic, we conducted a practical study on DOM-based Cross-site Scripting, a sub-class of XSS (see Sec. 7.1.1) that is caused by insecure JavaScript code. The remainder of this Chapter reports on our fully automated system to identify insecure JavaScript in real-world Web applications and the results of our study of the Alexa Top 5000 Web sites.

8.3.2 Technical background

Cross-Site Scripting is an attack in which an attacker is able to inject his own JavaScript code into a Web application, in such a way that the code is executed within a victim's browser in the context of the application. Since 2000, when one of the first XSS vulnerabilities was reported [48], novel attack variants were discovered. In 2005, Amit Klein published a paper in which he first mentioned the term *DOM-based XSS* and described the basic characteristics of this vulnerability [119]. In contrast to traditional (reflected and persistent) Cross-Site Scripting, DOM-based XSS is caused by incorrect client-side code rather than by server-side code. As described earlier, the dynamic nature of this client-side code makes it hard to detect or verify this kind of vulnerability.

In order to trigger a DOM-based XSS exploit an attacker is able to utilize a multitude of different attack vectors to inject his malicious payload (such as `location.href`, `document.referrer`, `window.name`, and many, many more). Depending on the Web application's program logic, it processes attacker-controllable inputs and at some point in time conducts a string-to-code conversion. As shown in our empirical study, this is a very common scenario. If input is not sanitized correctly, the attacker may be able to inject his own code into the application. Thereby, different subtypes of DOM-based XSS exist depending on the method used for converting the string to code:

HTML context Web applications commonly insert generated HTML code into the DOM via functions such as `document.write`, `innerHTML` or `insertAdjacentHTML`. When these functions are called, the browser parses the string parameter and interprets the contents as HTML code, which is then inserted into a certain position within the DOM. If user input flows into these sinks, sanitization or encoding functions have to be used in order to avoid code injection vulnerabilities. If the input is not sanitized correctly an attacker is able to inject own HTML tags including `<script>`, which enables JavaScript execution. For the specific differences between `innerHTML` and `document.write`, we refer the reader to Sec. 8.3.5.

JavaScript context Another commonly used method, which is sometimes vulnerable to DOM-based XSS, is the `eval` function. `eval` takes a string parameter, interprets it as JavaScript code and executes it. Besides `eval` and its aliases `setTimeout` and `setInterval`, there are also other contexts in which strings are converted into JavaScript code such as `script.innerText`, `script.text`, `script.textContent` and the assignment of strings to event handler attributes.

URL context If an attacker-controlled input flows into a URL attribute of any DOM node (such as `img.src`, `iframe.src`, `object. data` or `a.href`), an immediate conversion from a string to code does not occur. However, there are still several security problems related to this kind of flows. For example, if the attacker is able to control the complete URL, he could make

use of the `javascript:` or `data:` schemes to execute script code. If only parts of the URL are controlled, the attacker could still conduct redirects or phishing and in some cases even achieve JavaScript code execution as shown in Section 8.3.6.

Other contexts Besides those contexts that allow code execution, there are further sinks/contexts that are security sensitive such as `document.cookie`, the Web Storage API, postMessage or `setAttribute`. In Section 8.3.6, for example, we present a persistent DOM-based XSS vulnerability via `document.cookie`, which was discovered by our framework.

8.3.3 Approach Overview

In this chapter, we discuss a system to automatically detect and validate DOM-based XSS vulnerabilities. To address the outlined challenges in the assessment of client-side security problems, we decided to address the problem as follows: Instead of building analytical processes that complement [205] or emulate [177] the client-side behavior, we chose to integrate our techniques directly into a full browser.

More precisely, our system consists of two separate components: For vulnerability detection, we utilize a modified browsing engine that supports dynamic, byte-level taint-tracking of suspicious flows. Through directly altering the engine's string type implementation, we achieve complete coverage of all JavaScript language features and the full DOM API. We discuss the design and implementation in Section 8.3.4.

The second component is a fully automated vulnerability validation mechanism, that leverages the fine-grained context information provided by our taint-aware browsing engine. Due to the exact knowledge of data source and syntactical context of the final data sink, our system is able to create attack payloads that match the syntactic surroundings of the injection point. This in turn allows unambiguous vulnerability validation through verification that our injected JavaScript was indeed executed. This component is presented in Section 8.3.5.

8.3.4 Vulnerability detection: Modified Chrome

To automatically detect the flow of potentially attacker-controllable input (called a source) into a sink in the sense of DOM-based XSS, we decided to implement a dynamic taint-tracking approach. To ensure that edge-cases, which might not be implemented properly into pure testing engines like HTMLUnit, were to be properly executed, we chose to implement taint-tracking into a real browser. For this, we modified the open-source browser Chromium in such a manner that its JavaScript engine V8 as well as the DOM implementation in WebKit were enhanced with taint-tracking capabilities. For both components of the browser, we selected to use a byte-wise taint-tracking approach built directly into the respective string representations. In this fashion, we enabled our tool to not only distinguish between a completely untainted string and a string containing any potentially harmful content, but also to specifically get information on the origin of each given character in said string.

Labeling sources and encoding functions

To keep the memory overhead as small as possible, we chose to implement our approach in such a way, that information on a given character's source is encoded in just one byte. We therefore assigned a numerical identifier to each of the 14 identified sources (e.g. `location.href`, `location.hash` or `document.referrer`). Hence, we were able to encode this information into the lower half of the byte. To also be able to determine whether a given character was encoded using the built-in functions *encodeURI*, *encodeURIComponent* and *escape*, we used the lower three of the four remaining bits to store whether one or more of these functions were applied to the string. To represent a benign character, the lower four bits are set to 0.

Patching the V8 JavaScript engine

Google's JavaScript engine V8 is highly optimized in regards to both memory allocation and execution speed. Although the code is written in C++, V8 for the most parts does not make use of a class-concept using member variables when representing JavaScript objects like strings or arrays. Instead, a small header is used and objects components are addressed by only using given offsets relative to the object's address.

After careful examination of the given code, we chose to only encode the desired information directly into the header. Every object in V8 stores a pointer to its *map*. The map describes the class of an object. In V8, there are maps for each type of object. We found an used part of a bitmap in the maps and used it to create new map objects for tainted strings. Obviously, for strings of dynamic length, additional memory must be allocated to store the actual data. Based on whether a string is pure ASCII or also contains two-byte characters, this memory is allocated on creation of the object. The address of this newly created space is then written to one of the aforementioned offsets in the header. Along with the information that a string is tainted, we also need to store the taint bytes described above. To do this, we changed the string implementation such that additional *length* bytes are allocated. Since we wanted to keep the changes to existing code as small as possible, we chose to store the taint bytes into the last part of the allocated memory. This way, the functionality for normal access to a string's characters did not have to be changed and only functionality for taint information access had to be added.

As mentioned before, the V8 engine is optimized for performance. It therefore employs so-called generated code which is assembler code directly created from macros. This way, simple operations such as string allocation can be done without using the more complex runtime code written in C++. However, for our approach to easily integrate into the existing code, we chose to disable the optimizations for all string operations such as creation or sub-string access.

After patching the string implementation itself, we also instrumented the string propagation function such as `substring`, `concat`, `charAt`, etc. This is necessary to ensure that the byte-wise taint-tracking information is also propagated during string conversions.

Patching the WebKit DOM implementation

In contrast to the V8 engine, WebKit makes frequent use of the concept of member variables for its classes. Therefore, to allow for the detection of a tainted string, we were able to add such a member denoting whether a string is tainted or not. The string implementation of WebKit uses an array to store the character data. Hence, we added a second array to hold our taint bytes. Since strings coming from V8 are converted before being written into the DOM, we patched the corresponding functions to allow the propagation of the taint information. This is necessary because tainted data might be temporarily stored in the DOM before flowing to a sink, e.g. by setting the `href` attribute of an anchor and later using this in a `document.write`. To allow for correct propagation of the taint information, we not only needed to change the string implementation but also modify the HTML tokenizer. When HTML content is set via JavaScript (e.g. using `innerHTML`), it is not just stored as a string but rather parsed and split up into its tree structure. Since we want to ensure that taint information is carried into the tag names and attributes in the generated tree, these changes were also necessary.

Detection of sink access

Until now we discussed the tracking of tainted data inside the V8 JavaScript engine and WebKit. The next step in our implementation was to detect a tainted flow and to notify the user. Therefore, we modified all DOM-based Cross-Site Scripting sinks – like `document.write`, `innerHTML` or `eval`. We changed them in such a way that a reporting function is called each time a tainted string is passed to such a sink. In order to pass on the report to the user interface, we implemented a Chrome extension, that injects the JavaScript reporting function into the DOM. As such a function is callable from inside the runtime engine, we are able to report the flow to

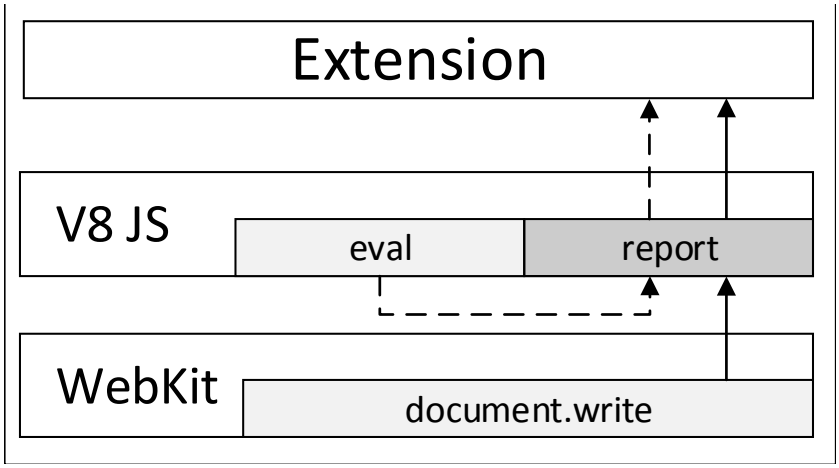


Figure 8.3: Report functionality

the extension. The details on the layout and implementation of this extension are presented in 8.3.6.

In WebKit's API used to provide access to the DOM tree for V8, the passed arguments are of V8's string class and are then converted to WebKit's string type. Hence, we chose to implement our reporting function into V8's string class, therefore allowing us to invoke it from the DOM API as well as directly from V8 using the provided string reference. When called, this function gathers information on the code location of the currently executed instruction and reports these alongside the taint information and details on the type of sink to the extension.

Figure 8.3 depicts this layout. Both the indicated functions `eval` and `document.write` use the reference to the passed string to invoke the reporting function which in turn passes on the information to the Chrome extension shown at the top.

8.3.5 Vulnerability verification: Automatic exploit generation

Although the taint-tracking engine delivers first indications for potential Cross-Site Scripting vulnerabilities, detecting a flow alone is not sufficient to ensure that a vulnerability was discovered. There are various reasons why a successful exploitation is not possible for an existing flow. For example, the Web site could use built-in or custom encoding or filter functions that are capable of defusing a malicious payload. Furthermore, other, random circumstance can occur that prevent an exploit from executing. For example, if the tainted value originates from a GET parameter, tampering with this parameter could trigger the Web server to load a different page or to display an error message in which the vulnerable flow is not present anymore. Therefore, a verification step is needed to tell vulnerable data flows apart from non-exploitable flows. In order to do so our system uses the data received from the taint- tracking engine to reliable generate valid Cross-Site Scripting Exploits. In this Section we describe the inner workings of the generation process.

Anatomy of a Cross-Site Scripting Exploit

To develop a system that is capable of generating valid XSS payloads, we first analyzed the nature of a Cross-Site Scripting exploit. In general, an exploit is context dependent. This means, that a payload, which an attacker seeks to execute, depends on how the Web application processes the attacker's input. So, if the input flows into the `eval` sink it has to utilize a different syntax than an exploit targeting flows into `document.write` (More details on context-dependent

exploit generation can be found in the Section 8.3.5). However, the structure of an exploit can be generalized to a non-context-dependent form.

Listing 1 shows two typical exploits. The first exploit targets a JavaScript context (e.g. `eval`), while the second one contains an exploit for an HTML sink (e.g. `document.write`). In many cases a tainted value was concatenated from several different strings, which are hard coded (benign/non-attacker-controllable) or coming from either one or more sources (attacker-controllable). Therefore, an attacker is only able to control parts of the string that flows into the sink. Immediate execution of JavaScript is often not possible at the location where the tainted/-controllable parts are inserted into the string/code (e.g. within quoted strings). Therefore, the exploit first has to break out of the current context to be able to execute the malicious script code. The first part of each exploit serves as a "break out sequence" to escape to a context where JavaScript execution is possible. In the cases presented in Listing 1 these sequences are "`');`" and "`"></a>`", respectively. Following the break out sequence, an arbitrary JavaScript payload or `<script>` tag can be executed. Afterwards, the exploit has to take care of trailing string fragments in such a way that these fragments do not interfere with the execution of the payload. For example, if a string that is passed to *eval* contains a syntax error, no code will be executed at all, even if the syntax error occurs at the very end of the string. To prevent this from happening an exploit has to include an escape sequence that renders trailing characters harmless. In the JavaScript case we simply comment out everything that follows our payload and in the HTML case we close the script block and include a `<textarea>` to interpret the rest of the string as simple text instead of HTML. To summarize our analysis, we conclude that a Cross-Site Scripting exploit takes the following generalized form:

$$exploit := breakOutSequence\ payload\ escapeSequence \tag{8.1}$$

In this, only the *breakOutSequence* and the *escapeSequence* are context-specific. While the *escapeSequence* is very trivial to choose, the *breakOutSequence* needs careful crafting to result in a successful exploit.

Listing 1 Example Cross-Site Scripting exploits

```
');alert('XSS');//
"></a><script>alert('XSS')</script><textarea>
```

Context-Dependent Generation of Breakout Sequences

After discovering a data flow from one or more sources to a sink, the taint-tracking engine delivers three pieces of information to the exploit generation framework:

1. Information on the the data flow (sources, sink, applied built-in filters)
2. Tainted value: the complete string that flowed into the sink (including benign and tainted parts from one or more sources)
3. Byte-wise taint information for each byte contained in the tainted string.

Based on the given sink the framework first determines the target context. Depending on this context, the tainted value and the taint information are passed to a context-sensitive *break out sequence* generation function. In the next step, the generator adds the desired payload and a context-specific fixed escape sequence. After constructing the exploit, the system builds a test case that can be executed in a completely automated fashion and reports back to the framework in case of successful exploitation.

HTML context-specific generation

An HTML context is present whenever a tainted string is directly converted into HTML code. This is the case for many DOM manipulation functions such as `document.write` or `innerHTML`.

As mentioned before, often only parts of a string may be tainted. Therefore, our system first determines the exact location of the tainted parts by analyzing the taint information. In order to create a valid exploit, the system needs to determine into which DOM node the tainted parts will be transformed when the string-to-HTML conversion takes place. In order to do so, the generator parses the complete string and identifies the corresponding nodes. Based on the node types the generator is able to plan the next step within the generation process. In this first step we distinguish between three different node types (See Listing 2 for examples):

1. **Tainted TagNode:** The tainted value is located inside an HTML tag. Either it is part of the tag name, an attribute name, an attribute value or a combination of those three possibilities.
2. **Tainted CommentNode:** The tainted value is contained within an HTML comment.
3. **Tainted TextNode:** The tainted value is placed outside of an HTML tag or in between a closing and an opening tag.

Depending on the the node type, break out sequences have to be generated differently. In the following, we explain the three different approaches:

TagNode generation If the tainted value is included within an HTML tag we first need to break out of this tag. Otherwise, opening a `<script>` tag would have no effect. If the tainted value is directly located within the tag, we can simple do so by adding a ">" sign to the break out sequence. If the tainted value resides within an attribute of the tag, the system first needs to determine the delimiter of the attribute. Most of the time such attributes are either enclosed by single or double quotes, however, sometimes, also no delimiter is present. So in order to break out of the tag in this case we need to add the delimiter of the attribute node before the angle brackets.

Now our payload is able to break out of the current (opening) tag and would be able to open a `script` tag, to execute the payload. However, some tags have special semantics for the text between the opening and the closing tag. So for example, HTML markup between an opening and closing `iframe` tag is only rendered in case iframes are not supported by the browser. Therefore, our generator optionally adds one or more additional closing tags at the end of the break out sequence for all present tags with special semantics. To summarize this, a TagNode break out sequences looks as follows:

$$TagNodeBS := [delimiter] > [closingTags] \tag{8.2}$$

Listing 2 Example Vulnerabilities

```
document.write('<script src="//example.org/'
            + taintedValue + '"></script>')

document.write('<div>' +taintedValue+ '</div>')

document.write('<!--' +taintedValue+ '-->')
```

CommentNode generation The generation of CommentNode break out sequences is very trivial in most of the cases. As comments in HTML do not have any special semantics for their content, we can simply break out of a comment by adding "- ->" to our break out sequence. However, such a comment could in rare cases be placed in between opening and closing tags of scripts, iframes, etc. So, again our system analyzes the string and adds closing tags for these elements if necessary. Summing up, a CommentNode break out sequence takes the following form:

$$CommentNodeBS := -->[closingTags] \tag{8.3}$$

TextNode generation Every character sequence that is placed outside a tag or a comment or that is located in between an opening and a closing tag is regarded as a TextNode by the HTML parser. In many cases executing a payload within a TextNode is straight forward. As we do not need to break out of the node itself, we can simply open a `script` tag and execute a payload. However, if the TextNode is placed between an opening and a closing tag of a `script` or `iframe` we again have to add closing tags if necessary.

$$TextNodeBS := [closingTags] \tag{8.4}$$

innerHTML vs document.write After we have generated the break out sequence for HTML context exploits, the system needs to choose a payload to execute. When doing so, some subtle differences in the handling of string-to-HTML conversion comes into play. When using `innerHTML`, `outerHTML` or `adjacentHTML` browsers react differently than `document.write` in terms of script execution. While `document.write` inserts `script` elements into the DOM and executes them immediately, `innerHTML` only performs the first step, but does not execute the script. So adding the following payload for an `innerHTML` flow would not result in a successful exploit:

```
<script>__reportingFunction__()</script>
```

However, it is still possible to execute scripts via an injection through `innerHTML`. In order to do so, the framework makes use of event handlers:

```
<img src="none_existent_resource" onerror="__reportingFunction__()">
```

When `innerHTML` inserts the `img` tag, the browser creates an HTTP request to the non-existing resource. Obviously, this request will fail and trigger the `onerror` event handler that executes the given payload. Depending on the sink we simply choose one of these two payloads.

JavaScript context-specific generation

JavaScript context-specific generation is necessary whenever a data flow ends within a sink that interprets a string as JavaScript code. This is the case for functions such as `eval` & `Function`, Event handlers (such as `onload` and `onerror`) and DOM properties such as `script.textContent`, `script.text` and `script.innerText`. While browsers are very forgiving when parsing and executing syntactically incorrect HTML, they are quite strict when it comes to JavaScript code execution. If the JavaScript parser encounters a syntax error, it cancels the script execution for the complete block/function. Therefore, the big challenge for the exploit generator is to generate a syntactically correct exploit, that will not cause the parser to cancel the execution. In order to do so, the system again has to determine the exact location of the tainted bytes.

Listing 3 shows a very simple vulnerable piece of JavaScript code. In the first step, the code constructs a string of benign/hard coded and tainted (`location.href`) parts. In a second step, it executes the code using `eval`. Thereby, this code can be exploited in slightly different ways. Either the attacker could break out of the variable `x` and inject his code into the function named test, or he could break out of the variable `x` and the function `test` and inject his code

into the top level JavaScript space. While the first method requires an additional invocation of the test function, the second exploit executes as soon as eval is called with a syntactically correct code. However, for the last case, the complexity of the break out sequence grows with the complexity of constructed code. Nevertheless, we do not want do rely on any behavior of other non-controllable code or wait for a user interaction to trigger an invocation of the manipulated code.

Listing 3 JavaScript context example

```
var code =  'function test(){' +
            'var x = "' + location.href + '";'
            //inside function test
            + 'doSomething(x);'
        + '}'; //top level
eval(code)
```

Therefore, we always seek to break out to the top level of the JavaScript execution. In order to do so, our system first parses the JavaScript string and creates a syntax tree of the code. Based on this tree and the taint information we extract the branches that contain tainted values. Listing 4 shows the resulting syntax tree for our example code and the extracted branch (in gray). For each of the extracted branches the generator creates one break out sequence by traversing the branch from top to bottom and adding a fixed sequence of closing/break out characters for each node. So in our example the following steps are taken:

1. FunctionDeclaration: ';'
2. FunctionConstructor: "
3. Block: '}'
4. Declaration: ';'
5. StringLiteral: '"'
6. Resulting Breakout Sequence: '";};'

To trigger the exploit we can simply construct the test case as follows: Based on the source (`location.href`), the system simple adds the break out sequence, an arbitrary payload and the escape sequence to the URL of the page:

```
http://example.org/#";};__reportingFunction__();//
```

When executed within a browser, the string construction process from Listing 3 is conducted and the following string flows into the `eval` call (Note: Line breaks are only inserted for readability reasons):

8.3.6 Empirical Study

As mentioned earlier, an important motivation for our work was to gain insight into the prevalence and nature of potentially insecure data flows in current JavaScript applications leading to DOM-based XSS. For this reason, we created a Web crawling infrastructure capable of automatically applying our vulnerability detection and validation techniques to a large set of real-world Web sites.

Methodology & Architecture Overview

To obtain a realistic picture on the commonness of insecure data flows that might lead to DOM-based XSS, it is essential to sample a sufficiently large set of real-world Web sites.

We designed our experiment set-up to meet these requirements, utilizing the following components: Our flow-tracking rendering engine to identify and record potentially unsafe JavaScripts

Listing 4 JavaScript Syntax Tree

```
FunctionDeclaration
  Identifier : test
  FunctionConstructor
    Identifier : test
    Block
      Declaration
        Identifier : x
        StringLiteral : "http://example.org"
    ExpressionStmt
      SpecialOperation : FUNCTION_CALL
        Reference
          Identifier : doSomething
```

```
function test(){
    var x = "http://example.org/#";
};
__reportingFunction__();
//doSomething(x);}
```

(as discussed in Sec. 8.3.4), our exploit generation and validation framework (as presented in Sec. 8.3.5), and a crawling infrastructure that automatically causes the browsing engine to visit and examine a large set of URLs.

Our crawling infrastructure consisted of several browser instances and a central backend, which steered the crawling process. Each browser was outfitted with an extension that provided the browser with the required external interface for communication with the backend (see Fig. 8.4). In the following paragraphs, we briefly document both the backend's and the extension's functionality.

Analysis engine: Central server backend

The main duty of the central analysis backend is to distribute the URLs of the examination targets to the browser instances and the processing of the returned information. The backend maintains a central URL queue, which was initially populated with the Alexa Top 5000 domains and subsequently filled with the URLs that were found by the browsers during the crawling process.

The browser instances transmit their analysis report and their findings to the backend. For each analyzed URL, analysis reports for several URLs are returned, as the browser instances not only check the main page but also all contained `iframes`. In our study, we received results for an average of 8.64 (sub-)frames for each URL that was given to a browser instance. After preprocessing and initial filtering, the backend passes the suspicious flows to the exploit generation unit (see Sec. 8.3.6).

Data collection: Browser Extension

As discussed in Section 8.3.4, we kept direct changes to the browser's core engine as small as possible, to avoid unwanted side effects and provide maintainability of our modifications. Our patches to the browser's internal implementation consisted mainly in adding the taint-tracking capabilities to the Javascript engine and DOM implementation. All further browser features

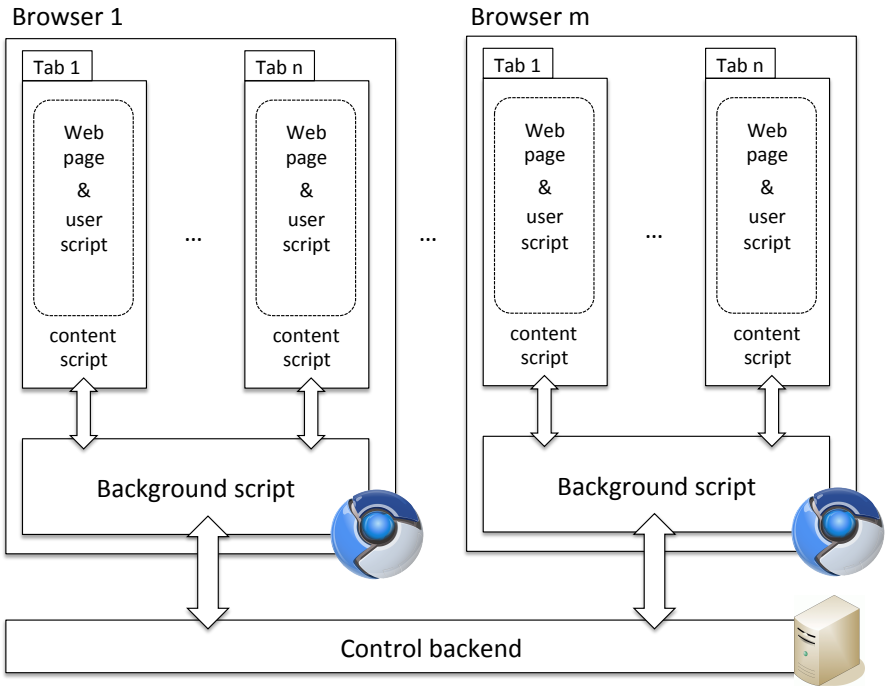


Figure 8.4: Crawling infrastructure

that were needed for the crawling and analyzing processes were realized in the form of a browser extension.

Following the general architecture of Chrome's extension model [86], the extension consists of a background and a content script (see Fig. 8.4). The *background script*'s purpose is to request target URLs from the backend, assign these URLs to the browser's tabs (for each browser instance, the extension opened a predefined number of separate browser tabs to parallelize the crawling process), and report the findings to the backend. The *content script* conducts all actions that directly apply to individual Web documents, such as collecting the hyperlinks contained in the page for the further crawling process and processing the data flow reports from the taint-tracking engine (see Sec. 8.3.4). Furthermore, the content script injects a small *userscript* into each Web document, that prevents the examined page from displaying modal dialogues, such as `alert()` or `confirm()` message boxes, which could interrupt the unobserved crawling process.

After the background script assigns a URL to a tab, the content script instructs the tab to load the URL and render the corresponding Web page. This implicitly causes all further external (script) resources to be retrieved and all scripts, that are contained in the page, to be executed. After the page loading process has finished, a timeout is set to allow asynchronous loading processes and script execution to terminate. After the timeout has passed, the content script packs all suspicious data flows, which were reported during execution of the analyzed page, and communicates them to the background script for further processing.

In addition to data flow and hyperlink data, the extension also collects statistical information in respect to size and nature of the JavaScripts that are used by the examined sites.

Observed Data Flows

As mentioned above, our initial set of URLs consisted of the Alexa top 5000. For each of these URLs we conducted a shallow crawl, i.e., all same-domain links found in the respective homepages were followed, resulting in 504,275 accessed Web pages. On average each of those Web document consisted out of 8.64 frames resulting in a final number of 4,358,031 (not necessary

	URL	Cookie	referrer	name	postMessage	Storage	Total
HTML	1,356,796	1,535,299	240,341	35,466	35,103	16,387	3,219,392
JavaScript	22,962	359,962	511	617,743	448,311	279,383	1,728,872
URL	3,798,228	2,556,709	313,617	83,218	18,919	28,052	6,798,743
Cookie	220,300	10,227,050	25,062	1,328,634	2,554	5,618	11,809,218
Storage	41,739	65,772	1,586	434	194	105,440	215,165
postMessage	451,170	77,202	696	45,220	11,053	117,575	702,916
Total	5,891,195	14,821,994	581,813	2,110,715	516,134	552,455	24,474,306

Table 8.8: Data flow overview, mapping sources (top) to sinks (left)

unique) URLs.

In total our infrastructure captured 24,474,306 data flows from potentially tainted sources to security sensitive sinks. Please refer to Table 8.8 for details on the distribution of flows, depicted by their sources and sinks.

Selective Exploit Generation

As shown in the previous Section, the total number of potentially vulnerable data flows from insecure sources to security sensitive sinks is surprisingly high. In our study, the sheer number of found flows exceeds the number of analyzed pages by a factor of about 48.5.

Both our exploit generation and validation processes are efficient. Generating and testing an XSS exploit for a selected data flow requires roughly as much time as the initial analyzing process of the corresponding Web page. However, due to the large amount of observed flows, testing all data flows would have required significantly more time than the actual crawling process. Hence, to balance our coverage and broadness goals, we selected a subset out of all recorded, potentially vulnerable data flows, based on the following criteria:

(C1) The data flow ended in a sink that allows, if no further sanitization steps were taken, direct JavaScript execution. Hence, all flow into cookies, Web Storage, or DOM attribute values were excluded.

(C2) The data flow originates from a source that can immediately be controlled by the adversary, without programmatic preconditions or assumptions in respect to the processing code. This criteria effectively excluded all flows that come from second order sources, such as cookies or Web Storage, as well as flows from the `postMessage` API.

(C3) Only data flows without any built-in escaping methods and data flows with non-matching escaping methods were considered. Data flows, for which the observed built-in escaping methods indeed provide appropriate protection for the flow's final sink were excluded.

(C4) For each of the remaining data flows we generated exploits. However, many flows led to the generation of exactly the same exploit payloads for exactly the same URL - e.g. when a web page inserts three scripts via `document.write` and always includes `location.hash` at a similar location. In order to decrease the overhead for testing the exploits, our system only validates one of these exploits.

Starting from initial 24,474,306 flows, we successively applied the outlined criteria to establish the set of relevant flows:

$$24,474,306 \xrightarrow{C1} 4,948,264 \xrightarrow{C2} 1,825,598 \xrightarrow{C3} 313.794 \xrightarrow{C4} 181,238 \tag{8.5}$$

Thus, in total we generated 181,238 test payloads, out of which a total of 69,987 successfully caused the injected JavaScript to execute. We discuss the specifics of these results in the next Section.

Found vulnerabilities

In total, we generated a dataset of 181,238 test payloads utilizing several combinations of sources and sinks. As discussed in Section 8.3.6 (C3), all flows which are encoded are filtered early on. For Google Chromium, which we used in our testing infrastructure, adhering to this rule we also must filter all those exploits that use either `location.search` or `document.referrer` to carry the payloads. This is due to the fact that both these values are automatically encoded by Chromium. Hence, we chose to test these vulnerabilities in Internet Explorer 10 whereas the rest of the URLs were verified using our aforementioned crawling infrastructure. Since the number of exploits utilizing `search` vulnerabilities amounts to 38,329 and the sum for `referrer` reached 5083, the total number of exploits tested in Chromium was reduced to 137,826, whereas the remaining 43,412 exploits were tested using Internet Explorer.

Out of these, a total number of 58,066 URLs tested in Chromium triggered our verification payload. Additionally, we could exploit 11,921 URLs visited in Internet Explorer. This corresponds to a success rate of 38.61% in total, and a success rate of 42.13% when only considering vulnerabilities exploitable in Chromium.

As we discussed earlier, we crawled down one level from the entry page. We assume that a high number of Web sites utilize content management systems and thus include the same client-side code in each of their sub pages. Hence, to zero in on the number of actual vulnerabilities we decided to reduce the data set by applying a uniqueness criterion. For any finding that triggered an exploit, we therefore retrieved the URL, the used break out sequence, the type of code (inline, eval or external) and the exact location. Next, we normalized the URL to its corresponding second-level domain. To be consistent in regards to our selection of domains, we used the search feature on alexa.com to determine the corresponding second-level domain for each URL. We then determined for each of the results the tuple:

$$\{\text{domain}, \text{break out sequence}, \text{code type}, \text{code location}\}$$

In regards to the code location, we chose to implement the uniqueness to be the exact line and column offset in case of external scripts and evals, and the column offset in inline scripts. Applying the uniqueness filter to the complete dataset including those pages only exploitable on Internet Explorer, we found a total of 8,163 unique exploits on 701 different domains, whereas a domain corresponds to the aforementioned normalized domain. Due to the nature of our approach, among these were also domains not contained in the top 5000 domains. Thus, we applied another filter, removing all exploits from these domains outside the top 5000. This reduced the number of unique exploits to 6,167, stemming from 480 different domains. In respect to the number of domains we originally crawled, this means that our infrastructure found working exploits on *9.6%* of the 5000 most frequented Web sites and their sub-domains.

When considering only exploits that work in Chromium, we found 8,065 working exploits on 617 different domains, including those outside the top 5000. Again filtering out domains not contained in the 5000 most visited sites, we found 6,093 working exploits on 432 of the top 5000 domains or their sub-domains.

Among the domains we exploited were several online banking sites, a poplar social networking site as well as governmental domains and a large internet-service provider running a bug bounty program. Furthermore, we found vulnerabilities on Web sites of two well-known AntiVirus products.

Selected Case Studies

During the analysis of our findings, we encountered several vulnerabilities which exposed interesting characteristics. In the following subsections, we provide additional insight into these cases.

JSONP + HTTP Parameter Pollution

As stated in Section 8.3, flows into URL sinks are not easily exploitable. Only if the attacker controls the complete string, he can make use of data and javascript URLs to execute JavaScript code. However, in our dataset we found a particularly interesting coding pattern, that allows script execution despite the fact that the attacker only controls parts of the injected URLs. In order to abuse this pattern a Web page must assign a partly tainted string to a script.src attribute that includes a JSONP script with a callback parameter (See Listing 5).

Listing 5 JSONP script include

```
var script = document.createElement('script')
script.src = "http://example.org/data.json?u="
        + taintedValue + "&callback=cb_name";
```

In many cases the callback parameter is reflected back into the script in an unencoded/unfiltered fashion. Hence, the attacker could inject his own code into the script via this parameter. However, the callback parameter is hard coded and the attacker is not able to tamper with it at first sight. Nevertheless, it is possible to inject a second callback parameter into the script URL via the `taintedValue`. This results in the fact that two parameters with the same name and different values are sent to the server when requesting the script. Depending on the server-side logic the server will either choose the first or the second parameter (We found both situations, and depending on the position of the `taintedValue` we were able to exploit both situations). Hence, by conducting this so-called HTTP Parameter Pollution attack, the attacker is able to inject his value into the content of the script, which is afterwards embedded into the Web page.

One particularly interesting fact is that simply encoding the `taintedValue` will not protect against exploitation. Instead, the JSONP callback parameter needs to be sanitized. During our experiments we found one vulnerable callback parameter quite often on many different Web sites, which seemed to stem from jQuery (or at least, always called the same jQuery function).

Broken URL parsing

As browsers sometimes auto-encode certain parts of user controlled values, it is not possible to inject code into some of the analyzed sources. One example for this is `location.search` that is auto-encoded by all browser except Internet Explorer. Another source that is encoded by every modern browser is `location.pathname`. An injection via `location.pathname` is in general not possible until the application itself decodes the value. An additional encoding or sanitization step is therefore not necessary for these values. This fact, however, also leads to security vulnerabilities when Web developers trust in this auto-encoding feature while at the same time conducting incorrect URL parsing.

In our analysis, we found many examples where this fact leads to vulnerabilities. In the following we cover some examples where code fragments seemed to extract automatically encoded values (and hence no sanitization is needed), but due to non-standard parsing, extracted also unencoded parts in malicious cases.

1. Task: Extract host from URL
2. What it really does: Extract everything between `www.` and `.com` (e.g. whole URL)
3. e.g. `http://www.example.com/#notEncoded.com`

```
var regex = new RegExp("/www\..*\.com/g");
var result = regex.exec(location.href);
```

1. Task: Extract GET parameter `foo`

2. What it really does: Extracts something that starts with `foo=`
3. e.g. `http://www.example.com/#?foo=notEncoded`

```
var regex = new RegExp( "[\\?&]foo=([^&#]*)" );
var result = regex.exec(location.href);
```

1. Task: Extract all GET parameters
2. What it really does: Last GET parameter contains the unencoded Hash
3. e.g. `http://example.com/?foo=bar#notEncoded`

```
location.href.split('?')[1].split('&')[x]
                           .split('=')
```

Persistent DOM-based XSS

As seen in Table 8.8, our system captured also some flows into cookies and into the Web Storage API. However, we did not include it into our automatic exploit generation. Nevertheless, we were able to manually find several persistent DOM-based XSS. We detected flows that first came from user input and went into Cookie or Web Storage sinks effectively persisting the data within the user's browser. In the cases where we could trigger a successful exploit, this data was then used in a call to `eval`, hence exposing the Web site to persistent DOM-based XSS.

window.name flows

Within our dataset, we detected a surprisingly high number (>2 million) of flows originating from `window.name` that we couldn't explain at first sight. Although some of them were exploitable, we soon discovered the reason for this number. Most of these flows are not exploitable via DOM XSS as they are caused by a simple programming error. When declaring a local variable a developer has to use the `var` keyword. If someone declares a variable named `name` inside a function and misses the `var` keyword or if a local variable is created directly within a script block that is executed in the global scope, the variable is declared global (See Listings 6 and 7). Since inside a browser, the global object is `window`, the data is written to `window.name`. If the same variable is used within a call to a sink within the same script block, the corresponding flow is not exploitable as window.name was overwritten with benign data. However, this fact represents another serious issue: `window.name` is one of the very few properties that can be accessed across domain boundaries. Hence, any data that is written to this property can be accessed by third parties. This programming error, therefore, represents a serious information leakage problem, if sensitive data is written to such a global `name` variable. Given the huge amount of flows, it is very likely that this pattern could be misused to steal sensitive information.

Effectiveness of Chromium's XSS Filter

Modern browsers like Chromium and its commercial counterpart Google Chrome are equipped with client-side filter capabilities aiming at preventing XSS attacks [35]. In order to analyze the effectiveness of Chromium's XSS Filter, we utilized our successful exploits and tried to execute them with the activated filter. Out of the 701 domains we found, 300 domains were still susceptible to XSS even with Chromium's auditor enabled.

After further examination, we found three distinguishing characteristics for these working exploits. For one, none of the exploits abusing JavaScript sinks, such as `eval()`, were detected by XSS Auditor. This stems from the fact that the auditor is implemented inside the HTML

Listing 6 window.name bug 1: Missing var keyword

```
function test(){
    name = doSomething();
    document.write(name);
};
```

Listing 7 window.name bug 2: Declaration within the global scope

```
<script>
    var name = doSomething();
    document.write(name);
</script>
```

parser and thus cannot detected direct JavaScript sinks. Furthermore, exploits that were caused by remote script includes were not detected. The third type of undetected exploits was caused by JSONP vulnerabilities as discussed in Section 8.3.6.

On a positive note, in our study, we found that none of the exploits that targeted inline vulnerabilities passed through the filter. However, please note, that this experiment carries no reliable indication of protection robustness in respect to the exploits, that were stopped. We did not make any attempts to obfuscate the exploit payload [95] or use other filter evasion tricks [99].

In 2011 Nikiforakis demonstrated that Chrome's filter is not able to cope with exploits that utilize more than one injection point at once [147]. If we take our figures from Section 8.3.6 into account, we see that a tainted string consists – on average – of about three tainted substrings. Thus, an attacker has on average three possible injection points in order to leverage the techniques presented by Nikiforakis. Therefore, we have good reasons to believe that the numbers presented in this Section must rather be seen as a lower bound.

8.3.7 Analysis of Survey Results

In this chapter, we documented a large scale empirical study on security aspects of client-side Web application code. Our study resulted in the identification of 6,167 unique vulnerabilities distributed over 480 domains, demonstrating that 9,6% of the Alexa top 5000 carry at least one DOM-based XSS problem.

Code coverage of the study

It has to be stressed at this point that this number has indeed to be interpreted as a strong **lower bound**, as the code coverage of the study is limited by several factors:

- Our system's crawlers did only a shallow analysis of the sites, restricted to the home pages and first level sub pages, leaving the majority of the site untouched.
- Due to the pre-filtering of the flows (see Sec. 8.3.6), only suspicious data flows that are immediately exploitable have received further processing. Other flows (e.g., through the `postMessage` API or via client-side storage) might yield in further insecurities contained in our data set.
- Our system conducted only a passive crawl. We conducted no interaction with the applications' UI or submitted any data (e.g., via HTML forms) to the sites. Thus, only a fraction of the sites' JavaScript was executed during the study.
- We only accessed the public parts of the sites. All portions of the corresponding Web applications that are only available to authenticated users are our of reach for the crawlers.

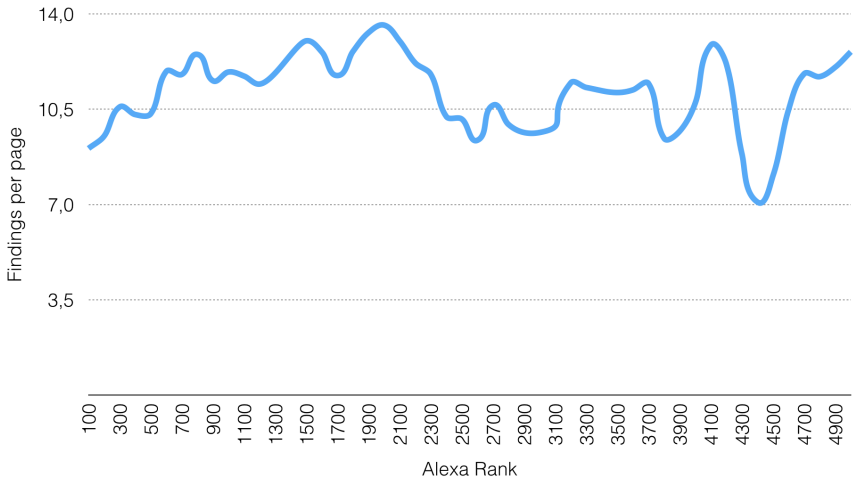


Figure 8.5: Average number of suspicious flows mapped against the sites' Alexa rank

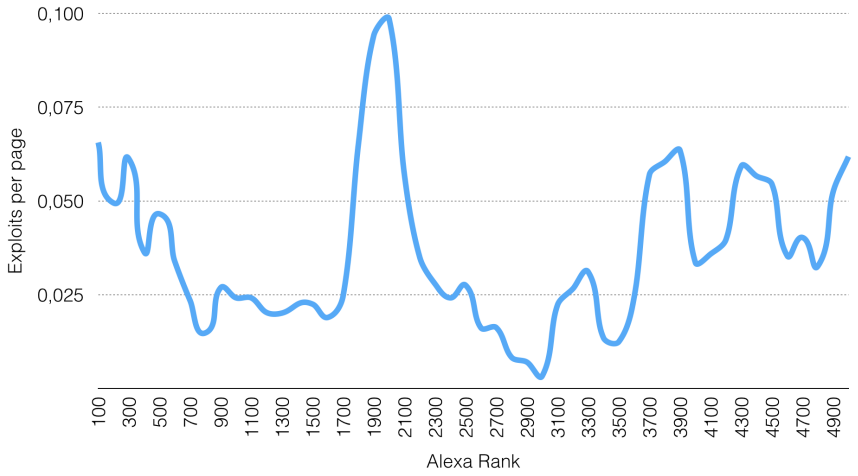


Figure 8.6: Average number of vulnerabilities mapped against the sites' Alexa rank

It can be expected that most non-trivial client-side code is utilized in realizing the actual application logic for the site's users. Thus, the potentially most complex (and, hence, most likely to be insecure) code has not been examined yet.

Hence, the effective number of existing vulnerabilities is most likely to be significantly higher than the reported numbers.

Site rank and region

In a secondary step, we examined if there are observable correlation between found complexity/insecurity of the surveyed code and a site's popularity or region of origin.

Popularity: To measure a site's popularity (which most likely should correlate to its engineering maturity and quality), we utilized the site's rank in the Alex ranking.

Region: Top level domains are no indicator for the regional origin of a Web site, as popular European sites frequently utilize international top level domains, such as `com`, `org`, or `net`. Hence, for all survey sites, we queried the geolocation of the site's Web sever's IP address.

Code complexity: As a measure for code complexity, we utilized the number of observed information flows, a number easily obtained using our taint-tracking engine. As site which deploys a lot of client-side code will expose a high number of information flows, while a site with limited client-side coding, will only account for few flows.

Using these indicators, we were able come to the following results:

- There is no apparent correlation between code complexity and site popularity (see Figure 8.5). In the Alexa Top 5000, the sites expose mostly uniform amount of suspicious information flows.
- Also, there is no clear correlation between site popularity (i.e., maturity) and existing vulnerabilities (see Figure 8.6). There is neither a "long-tail" phenomenon, in which the insecurity raises with declining of page rank, nor any other observable pattern.
- Same as it is with page popularity, there is no clear trend in visible in respect to code quality (according to average number of flows and numbers of found vulnerabilities) and a site's region of origin. Especially, taking the numbers for vulnerabilities, the two major western regions (EU and US) are very close to the general average.

Take aways

Even under the limiting circumstances in respect of achieved code coverage, we can conclude the following observations:

- **Unhandled client-side complexity:** The security implication of complex client-side JavaScript code are not properly handled by the developers of Web application, as we uncovered DOMXSS vulnerabilities in 10% of all surveyed sites. Given the comparatively well understand nature of XSS (opposed to novel threats such as Click/LikeJacking or PostMessage spoofing), this does not bode well for less well explored vulnerability classes.
- **Universal problem:** The observed problems apparently occur regardless of page popularity or origin.
- **Tip of the ice berg:** As mentioned above, by any means, the reported numbers are merely the lower bound of real client-side vulnerabilities. The real number (especially taking non-XSS issues into account) is most likely significantly higher.

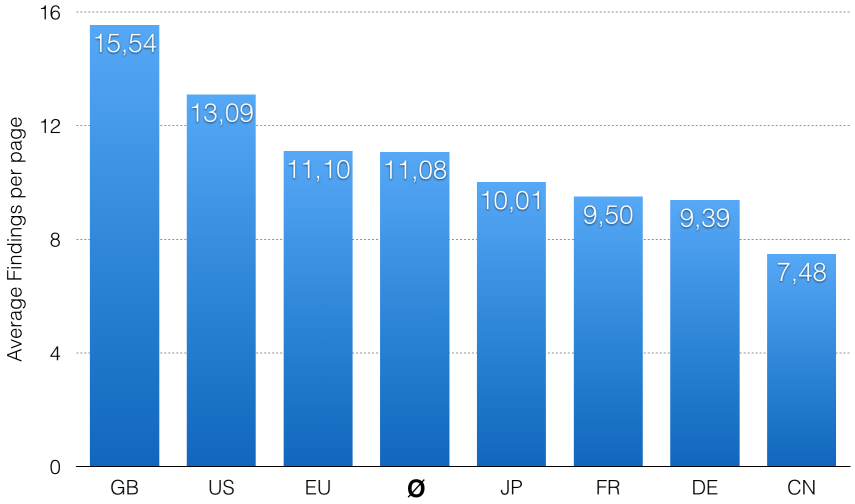


Figure 8.7: Average number of suspicious flows per examined region

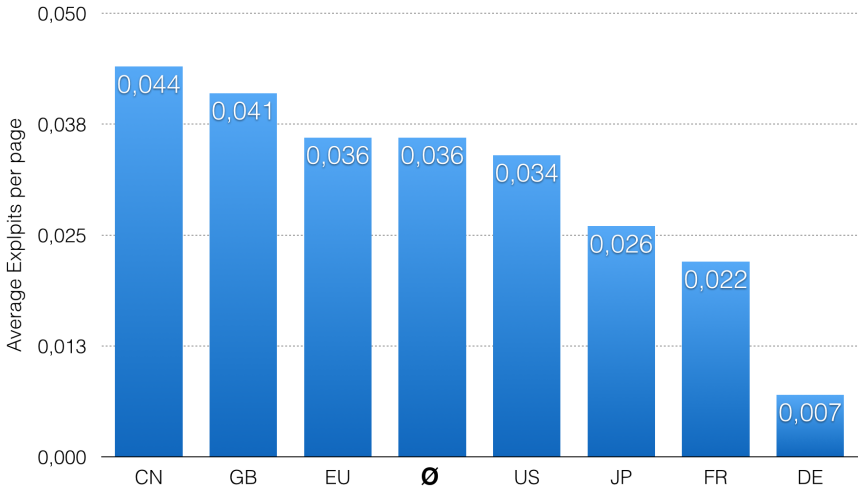


Figure 8.8: Average number of vulnerabilities per examined region

8.4 Large-scale Analysis of Third-party Security Seals

Nowadays, it has become rather uncommon for an entire month to go by without some news of a major security incident. Whether by bugs in cryptographic libraries [94, 196, 123], malware installed on points-of-sale terminals [121], the exploitation of web application vulnerabilities [122, 152, 169], or social engineering [27, 45], databases with credentials and personal details of millions of users seem to be finding their way into the hands of attackers, on a regular basis.

The monetary losses due to these events and due to cybercrime in general, are sizable. According to a recent report on the economic cost of cybercrime and cyber espionage, the cost of these activities reach 100 billion dollars each year, just for the United States alone [135]. Note that these losses are not just direct monetary losses, but also indirect ones. One interesting indirect loss is the opportunity cost to businesses due to reduced trust in online services.

Trust has always been a barrier to the adoption and use of new technology. In the context of the web, people are called to trust the websites of online companies, many of which do not have an offline brick-and-mortar presence, with their sensitive personal and financial information. In a 2008 survey, 75% of online shoppers did not like the fact that they had to provide their credit card and personal information online [104], while in a 2013 survey, only 61% of U.S. Internet users were banking online [84].

In such an environment of distrust, companies can distinguish themselves from others by not only securing their infrastructure but also convincing the user that they did so. This is even more so for smaller companies which do not enjoy the implicit trust afforded by the wide recognition of logos and household company names.

One way of achieving this goal is through the use of third-party security seals. A third-party security seal is an image that a website can embed in its HTML code which signals to consumers that the website has been scanned by a security company and has been found to be without issues, i.e., without vulnerabilities and without malware. Seals are typically provided by large security companies, like McAfee and Symantec, and are meant to be recognizable by the user and lend the credibility and trust associated with these large companies, to the certified site. Depending on the seal provider, security seals can cost anywhere between hundreds to thousands of dollars per year which can be a significant reoccurring security investment, especially for smaller companies.

Prior research on the topic of third-party security seals has mostly focused on whether a user recognizes their presence on certified websites and whether they result in higher confidence and thus increased sales [37, 93, 106, 116].

In [201], we report on a three-pronged analysis we have performend on third-party security seal providers. Instead of measuring whether users trust security seals, we want to know whether these seals *should* be trusted in the first place. First, we compile a list of ten popular seal providers and analyze their certification methods. We discover that, among others, the security checking of seal-using websites is done almost entirely in a black-box fashion where the seal providers try a list of automated attacks on their clients. By designing and deploying a known vulnerable web application, we witness the inaccuracy and haphazardness of these scanners, with the best one detecting less than half of the known vulnerabilities.

Second, we turn our attention to existing businesses that use third-party security seals, discovering more than 8,000 websites certified to be secure by the studied seal providers. By gathering and comparing features that are telling of a website's security hygiene, e.g., the use of HttpOnly cookies and Anti-CSRF tokens in HTML forms, we show that seal-using websites are *not* more prudent than other websites of an equivalent nature and popularity. Moreover, by obtaining explicit permission for a manual penetration test on nine seal-using websites, we demonstrate how a moderately motivated attacker can discover high-risk vulnerabilities in most certified websites, in less than one working day.

Finally, taking into account the workings of seal providers, we detail a series of attacks on websites that are actually facilitated by the use of third-party security seals. Among others, we describe the architecture of a completely passive vulnerability oracle that allows an attacker to discover easily exploitable websites by monitoring the appearance and disappearance of third-

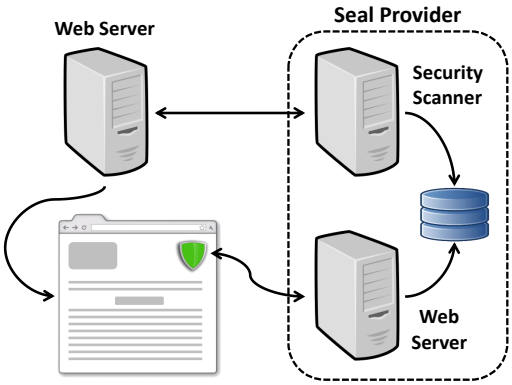


Figure 8.9: High-level view of the architecture and delivery of third-party security seals

party security seals on seal-using websites.

8.4.1 Security Seals

In this section, we describe the general workings of third-party security seals and list the ten seal providers that we analyzed, together with their features and deviations from the generic model of a third-party security seal.

General Architecture

In general, third-party security seals follow the architecture shown in Figure 8.9 which is comprised of two distinct components. We arrived at this architecture through the analysis of all ten investigated providers, and the recording of common features.

First, if seal providers are to give meaningful security certifications to other websites, they must have a way to assess the security stance of those websites. This is done through the use of an automated scanner which periodically checks the websites of their clients for the presence of security issues, usually involving vulnerabilities and, in some cases, malware. While the exact scanning frequency depends on each seal provider and the version of the seal a customer has bought, the scans are usually repeated either on a daily or a weekly basis. The clients of seal providers are given access to a web-based "control panel" where they can inspect the findings of the latest security scan. If the scan discovered vulnerabilities, the client is given a description of the vulnerability, its exact location and nature (e.g. reflected XSS found on parameter `page` of `index.php`), and some generic guidance for its remediation, usually in the form of links towards whitepapers and other security resources. When a vulnerability is discovered, the owners of certified websites are typically given a "grace period" of a few days which they can use to correct the discovered vulnerability before it affects their certification, i.e., the visual component displayed on their website.

The second component of a seal-provider is the seal itself. When a website owner subscribes her website for a third-party security seal, she is given a snippet of HTML code (or JavaScript code which creates an HTML snippet) which she is instructed to place in her page, at the position where she wants the third-party seal to appear. The general format of seal snippets is as follows:

```
<a href="https://seal-provider.com/info?
        client=example.com">
 <img src="https://seal-provider.com/shield?
        client=example.com"/>
</a>
```

#Clients	Name	Yearly Cost	Vulnerability Scan	Malware Scan	Server-side access	Server AuthN	Disappearing seal	Grace Period (Days)
3,980	Norton Secured	$995	✓	✓	–	–	–	NA
3,029	McAfee SECURE	$300	✓	✓	–	–	✓	Unknown
463	Trust-Guard	$697	✓	–	–	–	✓	7
459	SecurityMetrics	$120	✓	✓	–	–	–	0
281	WebsiteProtection (GoDaddy)	$84	✓	✓	✓	–	✓	0
118	BeyondSecurity	$360	✓	–	–	✓	✓	0
109	ScanVerify	$100	✓	–	–	–	✓	2
68	Qualys	$495	✓	✓	–	–	✓	3
48	HackerProof	$2,295	✓	–	–	–	✓	5
4	TinfoilSecurity	$2,388	✓	–	–	✓	–	NA

Table 8.9: Overview of evaluated seal providers and their features

This snippet always involves an HTML image tag which dynamically requests an image from the web server of the seal provider. If a website is found to be secure, the server will respond with an image of a seal, as shown in Figure 8.9, which typically includes the logo of the seal provider and is meant to be recognizable by the visitors of the seal-using website and lend it the credibility associated with the seal-provider. Usually, the image is wrapped in an anchor tag which is clickable, leads to the domain of the seal provider, and shows the visitor of a certified website more information about the seal and the seal provider, e.g., the coverage of the scan that produced the seal, and the day of the last scan. Apart from providing more information, this linked page creates an obstacle for website owners who wish to create the illusion of being certified, e.g., by showing a fake security seal on their website, possibly scraped by the website of a seal provider's paying client. Since the extra information is provided in a page under the seal-provider's domain, a non-paying user cannot mimic that part of the certification.

Interestingly, for the majority of seal providers, when a vulnerability is discovered and the certified website fails to correct it within its allotted grace period, the security seal merely becomes invisible. That is, even when a website is vulnerable, a visitor of that website will never see a "negative" security seal. The only way that a visitor can discover this fact is by the examination of the HTML source code of a page and the discovery of the aforementioned HTML snippet – a task well out of reach of common users of the web. As such, the vast majority of web users are not able to distinguish between a website that does not use a third-party security seal, and one that does but is vulnerable.

Seal Providers

To discover third-party seal providers, we searched in a popular search engine for phrases such as "site security seal" and "site safety seal". For each result, we manually examined the website of each seal vendor to ensure that the seal coverage included the detection of vulnerabilities, as opposed to other types of seals that verify a site's identity and the proper use of SSL. Due to the extensive labor involved with the evaluation of each seal provider and its clients, we limited ourselves to ten providers of security seals.

Table 8.9 lists the ten investigated third-party security seal providers ordered by the number of their clients that we could discover (we describe the process of client discovery in Section 8.4.2). One can see that the services vary greatly in terms of popularity as well as in terms of cost. Interestingly, there seems to be no correlation between the popularity of a seal provider and the price of seal. Since the two, by far, most popular seal providers are also two large, recognizable antivirus companies, it is likely that the popularity of seal providers is related more to brand recognition and less to other factors, such as price and word-of-mouth.

Five out of the ten seal providers support malware scanning in addition to vulnerability scanning, yet only one provides the option of scanning the server for malware over the FTP

protocol. As such, the other services can only discover malware on the indexable pages and directories of a website. Only two out of the ten investigated services have the option of server authentication, i.e., scanning the part of a website that is behind a registration wall. In these two cases, website owners can give the credentials of a user to the seal provider and the location of the login page. This means that the vulnerability scanners of the majority of seal providers will not be able to find vulnerabilities and malware that reside on the authenticated part of a website.

The investigated seal providers exhibit varied behavior when it comes to how they react in the presence of a discovered vulnerability. The majority of seal providers returns an invisible image when the client fails to mitigate the vulnerability within the grace period. However, there are two types of deviation from this behavior: the Norton Secured seal will always be shown, even when vulnerabilities were found, and similarly, for SecurityMetrics and TinfoilSecurity, the seal will also remain visible, but the status page on the seal provider website will no longer show that there is a passing certification.

Finally, we would like to stress that although security seals have been around for more than a decade, it is still a market that is actively being developed. During the course of our research, McAfee and Godaddy rebranded their security seal products. The McAfee SECURE seal, which previously offered both a malware scan and vulnerability analysis as a single package, was split and now only offers a security seal for passing the malware scan. In our research, we evaluated the combination of the malware scan and vulnerability scan, as was originally offered. WebsiteProtection, a GoDaddy product, was rebranded as SiteLock, which most likely reflects a change in the third-party that GoDaddy relies on for their security seal product. As in the case of the McAfee security seal, our research is mainly based on the original product that was offered by GoDaddy.

8.4.2 Adoption

As explained in Section 8.4.1, when website owners subscribe their websites to seal providers, they are given a small HTML snippet that they have to include in their websites. This snippet is responsible for fetching and displaying the security seal to the visitors of the certified site. In this section, we take advantage of this snippet in order to automatically detect seal-using websites, in an effort to understand the nature of websites that choose to certify themselves using security seals.

More specifically, using a crawler based on PhantomJS, we performed a shallow crawl of the top 1 million websites according to Alexa, searching for inclusions of specific remote images and anchor tags from each of the ten studied seal providers. To account for seal-using websites that are not part of the top 1 million Alexa websites, we used some advanced search features of Google's search engine, which we based on the same HTML snippets. For example, the following search query `site:scanverify.com/siteverify.php` returns a list of websites using seals by ScanVerify. Using these processes we were able to discover a total of 8,302 seal-using websites.

From the 8,302 websites, 73.64% was part of Alexa's top 1 million websites ranking. Figure 8.10 shows the distribution of these websites across the ranks of Alexa. The distribution is right-skewed where the usage of third-party security seals decreases together with the ranking. Our findings indicate that websites that are already popular, still choose to use seals as a mechanism of convincing users that they do take security seriously.

To identify the nature of these 8,302 websites, we used TrendMicro's public website categorization database [198]. Figure 8.11 shows the ten most popular categories of seal-using websites. As one can see, the "Shopping" category is by far the most popular with 35.74% of the entire dataset being categorized as such. Given the motivation for using third-party security seals that the seal providers themselves use, this result makes intuitive sense. Security seals are advertised as capable of increasing a user's trust for any given website which, in turn, translates to increased sales. As a matter of fact, most seal providers' advertising campaigns heavily rely on testimonials where existing clients claim to have seen a significant increase of sales (in the range

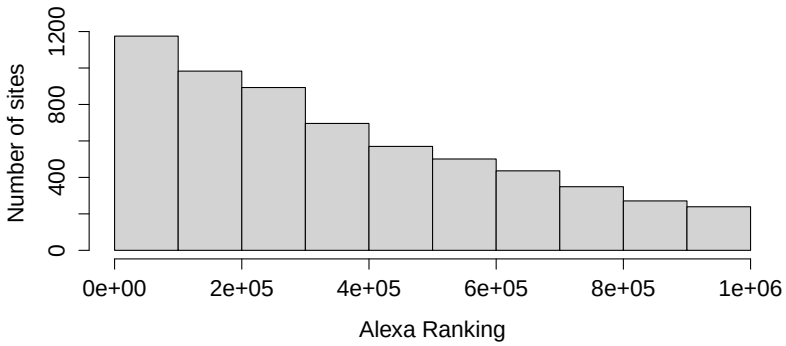


Figure 8.10: Distribution of seal-using websites in the Alexa top 1 million websites

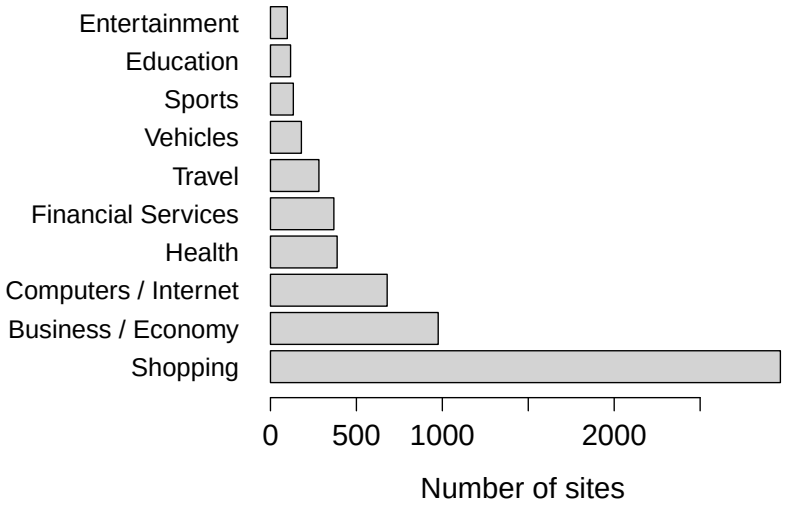


Figure 8.11: Ten most popular categories of seal-using websites

of 5%-20%) after the adoption of their security seal. As such, shopping websites are the primary target audience for buying security seals from seal providers. This result is also interesting from a security point of view. Websites belonging to e-shops are highly dynamic in nature, with frontends and backends, and various e-commerce modules. As such, their extended attack surface, together with the prospect of exfiltrating financial and personal data upon compromise, make shopping websites more attractive targets over other categories of websites, such as blogs and news websites.

8.4.3 Security Evaluation

Seal vendors claim that a security seal increases the trustworthiness of a certified website and leads to increased sales. In this section, we seek to understand whether a user's trust *should* increase in the presence of a third-party security seal.

In order to assess the thoroughness of service provided by the ten investigated seal providers, we conducted the following three experiments. First we compare the security practices of seal-using websites to other equivalent websites which do not make use of seals. Second, by obtaining permission for penetration testing, we investigate whether a moderately interested attacker would be able to find a vulnerability in a supposedly secure website, i.e., a website bearing a security seal. Third, we set up a webshop including multiple vulnerabilities, in order to understand which

vulnerabilities are discoverable by seal providers and how easy it is for a vulnerable site to obtain a clean bill of health.

Comparison to non-certified websites

When a website uses a third-party security seal, it may seem reasonable to assume that the administrators of that website are, in general, interested in the security of their site and are thus taking all the necessary precautions to secure their services and protect their users.

In this section, we test this assumption by comparing the adoption of popular security mechanisms by seal-using websites against the adoption of the same mechanisms by equivalent websites which do not use third-party seals. Next to security mechanisms we also test for issues that can be detected in a non-intrusive way.

Comparison Dataset To have meaningful comparisons between seal-certified and non-certified websites, the second set of websites must be similar to the first one, in all matters except for the adoption of a security seal. While this is virtually impossible to establish from the client-side without full knowledge of an application's codebase and environment, we provide an approximate solution as follows.

For every seal-utilizing website, we attempt to identify another site of the same category within ten places in the Alexa ranking of the top 1 million websites. If, for instance, `buyfromhome.com` is a seal-utilizing e-shop ranked at the 100th place, we search for another e-shop site either ten ranks above, or ten ranks below the 100th place. The direction of our search is decided probabilistically using the probability distribution of a fair coin. As before, the categories of each site are determined using TrendMicro's public website categorization engine [198].

Using this process we were able to match 2,238 seal-using websites to 2,238 other websites of equivalent rank and category that do not use third-party security seals.

Security Indicators In recent years, as a response to a continuous battery of exploitations of web application vulnerabilities, browser vendors and the research community introduced a series of client-side security mechanisms that are today available in the vast majority of modern browsers. These client-side mechanisms are usually guided by server-side policies, where the web server expresses its security desires through HTTP headers and the browsers are responsible for enforcing them at the side of the user. In addition, web application programmers have come up with various "best practices" that should always be followed, e.g., an anti-CSRF token in all forms.

While the presence or absence of such security mechanisms does not equate to proof of the presence or absence of exploitable vulnerabilities, they still can be used as indicators of the *security hygiene* of any given website. In addition, these mechanisms can be detected in a completely passive fashion thereby not incurring any unnecessary stress on web applications. In prior research, Nikiforakis et al. [149] combined some of these indicators in a Quality-of-Maintenance metric and applied it on servers offering remote JavaScript libraries. Vasek and Moore investigated whether certain server characteristics can be used as risk factors for predicting server compromise [203], and found that HttpOnly cookies can, for some types of compromise, be negative risk factors, i.e., their presence is correlated more with non-compromised websites rather than compromised ones.

For our experiment, we searched for the presence or absence of the following, passively discoverable, security mechanisms and best practices:

- HTTP Strict Transport Security (HSTS)
- Secure Cookies
- HttpOnly Cookies
- Content Security Policy (CSP)

Security Mechanism or Issue	Sites w/ Seals (%)	Sites w/o Seals (%)	Significantly different? (p-value)
HSTS	1.05	1.06	✗ (1.00)
Secure Cookies	1.83	0.42	✗ (0.06)
SSL Stripping	15.45	15.64	✗ (0.99)
X-Frame-Options	3.71	5.14	✓ (0.02)
HTTP-Only Cookies	42.27	29.98	✓ (<0.01)
CSP	0.00	0.00	– (NA)
Anti-CSRF Tokens	6.39	11.89	✓ (<0.01)
X-Content-Type	0.00	0.00	– (NA)
iframe sandbox	0.18	0.04	✗ (0.37)

Table 8.10: Comparison of the discovered issues and security mechanisms between websites with and without seals. Highlighted entries denote the better value, in the statistically significant cases.

- X-Frame-Options (XFO)
- iframe sandboxing
- Anti-CSRF Tokens
- X-Content-Type-Options
- SSL-stripping Vulnerable Form

Results For each of the 4,476 websites, we used a crawler based on PhantomJS to automatically visit a site's main page and ten other pages within the same domain, which were obtained using a shallow crawl. To simplify comparisons, we counted the present security mechanisms and issues in a binary fashion. If, for example, one page of the domain `example.com` used an HttpOnly cookie, then we credited the entire `example.com` domain with that security mechanism.

Table 8.10 shows the percentage of security mechanisms and issues discovered in the seal-utilizing websites and in our gathered set of equivalent websites. To compare the adoption between our two sets of websites, we conducted a two-sided hypothesis test for the comparison of two independent proportions. For each row in Table 8.10, our null hypothesis (H_0) was that the true proportion of that mechanism is the same between the two sets, while the alternative hypothesis (H_A) was that the true proportions of that mechanism for the two sets are *different* from each other. The last column of Table 8.10 shows the results of each hypothesis test, i.e., whether the adoption of each mechanism is different in a *statistically significant* way, and reports the p-value of the hypothesis test. Following standard practices in hypothesis testing, 0.05 was the cut-off point for the computed p-value, over which the null hypothesis is maintained.

As one can see, only one third of the measured proportions were different in a statistically significant way, while in two out of the three cases, the difference was credited in favor of the websites *without* security seals. The lack of more significant differences between the adoption of security mechanisms can be interpreted as a lack of systematic difference between the security hygiene of seal-using websites and the hygiene of equivalent websites that do not use seals. This, in turn, hints towards the absence of a holistic security strategy by the adopters of third-party security seals. In other words, if the intuitive notion that bad security hygiene is correlated with increased probability of compromise is true, seal-using websites are *not* more secure than their non-seal equivalents.

Penetration Testing

Security seal providers claim to protect websites by periodically scanning for the presence of thousands of vulnerabilities. One could assume that this would result in the detection of most easily discoverable vulnerabilities, thus making the discovery of a vulnerability by an attacker, a long and arduous task.

To test this hypothesis, we contacted 1,000 seal-using websites asking for explicit permission to conduct a manual penetration test. While the vast majority of websites never responded to our request, nine websites granted us permission to proceed. In order to avoid bias in our experiment we simulated a moderately motivated attacker who only has eight hours (one working day) to find a vulnerability before proceeding to the next target.

In our manual penetration test, we evaluated websites for several common vulnerabilities, such as SQL injection, cross-site scripting and CSRF, as well as for application-logic attacks. Despite the limited time available for evaluating each website, we were able to find several severe vulnerabilities, such as XSS and SQL Injection, for seven out of the nine evaluated websites. Additionally, we found HTTP parameter tampering vulnerabilities, as well as application-flow vulnerabilities, in three out of the four webshops that we analyzed, allowing us to order products and services for arbitrary prices.

These results clearly indicate that the hypothesis that attackers with limited resources are unable to find vulnerabilities in sealed websites, is false. In our manual analysis, we could register on the websites under evaluation, access content that requires authentication, and reason about prices in shopping carts – actions that are not supported by the automated scanners of most seal providers. Besides these general limitations that render seal providers unable to find vulnerabilities which require a series of coherent actions, we also found easily discoverable vulnerabilities, which were missed by the seal providers, in six out of nine websites. These vulnerabilities consist of cross-site scripting vulnerabilities where a GET or POST parameter was reflected without proper encoding, and even a "textbook" SQL injection.

We also want to mention one incident that demonstrates the haphazardness of the certification of certain seal providers. In one specific case, an e-shop, certified to be secure by a third-party seal provider, gave us an SQL error while contacting them to ask for permission for a manual penetration test. When reviewing the case, we realized that we had used a contraction in our message to the website where we inadvertently introduced a single quote, e.g., the phrase "do not" contracted to "don't". The SQL error was generated by that single quote in the message body.

Vulnerable Webshop Experiment

Being able to accurately assess the state of security of a website, is one of the key requirements of a seal provider, if the seal is to be trusted by consumers. That is, if trivially vulnerable websites can be certified as secure, then the certification becomes void of meaning. To evaluate the accuracy of the tools used by seal providers to verify a website's security, we set up a webshop which included a number of severe vulnerabilities. More specifically, we used an outdated version of PrestaShop, a popular open source e-commerce application, which suffers from a cross-site scripting vulnerability, and expanded the attack surface by including several other vulnerabilities spanning many typical web application vulnerability classes.

The vulnerable webshop reflects a realistic website containing vulnerabilities we encountered in real-world websites during our penetration testing. Next to a number of well known vulnerabilities, such as SQL injection and XSS, we also included vulnerabilities that are less popular, such as a remote JavaScript inclusion from a stale domain and a CSRF issue with OAuth login. In total, the following twelve vulnerabilities (V_1 to V_{12}) were present in our vulnerable web application:

- **SQL Injection (V_1):** A textbook example of SQL injection. One of the GET parameters was not properly sanitized and a user-controlled SQL statement could be executed.

- **SQL Injection - Ajax (V_2):** Similar to V_1, but only Ajax requests were made to this endpoint. Consequently, this endpoint could only be discovered by executing JavaScript code.
- **Sensitive files (V_3):** We uploaded a `phpinfo.php` file, which discloses sensitive information of a PHP installation, and a `.git` folder, which can leak the contents of several sensitive files.
- **Stale remote JavaScript inclusion (V_4):** On each page, a JavaScript file was included from an unregistered domain. An attacker could register that domain, and serve malicious JavaScript from it, which would be executed on all pages of the vulnerable webshop.
- **OAuth - CSRF parameter (V_5):** A "Login with Facebook" link was added, which points to Facebook's OAuth endpoint. This link did not contain a `state` parameter, which would allow an attacker to perform a CSRF attack and make a victim log in as the attacker.
- **Malware (V_6):** Every page of the webshop contained a link to a malicious executable. This link was made invisible using CSS in order to prevent infecting casual visitors of our test website.
- **Directory listing (V_7):** One of the directories that stored images, allowed directory listing. This directory also contained malicious executables.
- **Reflected XSS (V_8):** A GET parameter on one page of our web application was reflected without any encoding.
- **Reflected XSS - form action (V_9):** Similar to V_8, but the endpoint was located in the `action` attribute of a `form` element, as opposed to the `href` attribute of a link as in V_8.
- **Reflected XSS - additional parameter (V_{10}):** For this vulnerability, the query parameters were reflected without encoding. This only happened when the `controller` parameter was set to a certain value, so a GET parameter needed to be added for the discovery of this vulnerability.
- **Reflected XSS - JavaScript context (V_{11}):** In this case, again a GET parameter was reflected in a page, but the "`<`" and "`>`" characters were properly encoded. However, since the parameter was reflected in a JavaScript string context, an attacker could terminate the string with a quote, which was not escaped, and inject arbitrary JavaScript code.
- **DOM-based XSS (V_{12}):** On several pages, the fragment of the URL (the portion after the `#` sign in the URL), was written to the document, without any encoding.

From the ten seal providers listed in Table 8.9, we were able to purchase seals (or get free trials) from eight of them. The two missing providers implemented strict checks for the existence of a valid business which we did not attempt to bypass.

Table 8.11 shows the vulnerabilities that were discovered by each seal provider. The names of the seal providers have been anonymized, as our study is not meant to promote one product over another, but rather to show the coverage, or lack thereof, by all companies. The first thing that one can notice is that *all* seal providers found less than half of the vulnerabilities. Even more worrisome is that two seal providers did not manage to find any vulnerabilities. By analyzing the requests made to our web server, we found that these seal providers merely ran Nmap and Nessus scans, which listed open ports and checked for the presence of certain files. These scanners are not meant to discover vulnerabilities in web applications, thus they are far from sufficient to evaluate the security of a website.

Seal provider	Vulnerabilities						
	V_1	V_2	V_3	V_4	V_5	V_6	V_7
Seal Provider 1	–	–	–	–	–	N/A	–
Seal Provider 2	–	–	–	–	–	N/A	–
Seal Provider 3	–	–	✓	–	–	✓	✓
Seal Provider 4	–	–	–	–	–	✓	–
Seal Provider 5	✓	–	–	–	–	N/A	–
Seal Provider 6	✓	✓	–	–	–	–	–
Seal Provider 7	✓	✓	–	–	–	–	–
Seal Provider 8	✓	–	✓	–	–	N/A	–

	V_8	V_9	V_{10}	V_{11}	V_{12}	Coverage
Seal Provider 1	–	–	–	–	–	0/11 (00.0%)
Seal Provider 2	–	–	–	–	–	0/11 (00.0%)
Seal Provider 3	–	–	–	–	–	3/12 (25.0%)
Seal Provider 4	✓	✓	–	–	–	3/12 (25.0%)
Seal Provider 5	✓	✓	–	–	–	3/11 (27.3%)
Seal Provider 6	✓	✓	–	–	–	4/12 (33.3%)
Seal Provider 7	✓	✓	–	–	–	4/12 (33.3%)
Seal Provider 8	✓	✓	✓	–	–	5/11 (45.5%)

Table 8.11: Vulnerability detection results. For each seal provider, the vulnerabilities that were discovered are indicated with a checkmark.

V_8 and V_9, the two "standard" reflected XSS vulnerabilities, were found by the majority of seal providers. Even though V_1 was a textbook example of SQL injection, only half of the seal providers managed to find this vulnerability. Moreover, only Seal Provider 6 and 7 were able to find the SQL injection vulnerability in the Ajax endpoint (V_1). Interestingly, these were also the only two seal providers whose scanners executed the JavaScript code of our web application. Since more and more websites rely on JavaScript for delivering functionality to end users, a lack of support for JavaScript will certainly limit the number of vulnerabilities that a vulnerability scanner is able to find.

From the four seal providers that claim to check for the presence of malware, only two managed to find the malware present on our vulnerable web application. Note that we purposefully uploaded malware that was detected as such by the vast majority of antivirus engines on VirusTotal, to ensure that the malicious executables would be flagged by any proper antivirus product. Interestingly, one of these seal providers only managed to find the malware binary *after* they were given FTP access to our server, an optional feature they provide. The inability to find publicly-reachable malware by browsing our webshop, is another indication that the security-scan employed by seal providers is incomplete.

Overall, it is clear that the coverage of security seals leaves much to be desired. Even if a website would employ multiple security seals, certain classes of vulnerabilities would still remain fully undetected. While one can argue that some of the vulnerabilities present in our webshop are of a more exotic nature, such as V_4 and V_5, the fact is that these vulnerabilities are known by the security community, and can be easily discovered by an automated scanner as long as the detection logic is present. The absence of support for them, indicates that automated scanners may have trouble detecting newer vulnerabilities, even if those are easier to detect than traditional ones.

Finally, we compared the coverage of the seal providers with that of three popular web application vulnerability scanners. Web application vulnerability scanners typically work in a fully automated way and, despite their shortcomings [66], are a popular option for discovering vulnerabilities in websites. As Table 8.12 shows, the coverage of the scanners is higher than all but one of the tools employed by the security seal providers, which again shows that the scans

Vulnerability Scanner	Vulnerabilities						
	V_1	V_2	V_3	V_4	V_5	V_6	V_7
Acunetix	–	✓	–	–	–	N/A	✓
HP WebInspect	–	✓	–	–	–	N/A	✓
Burp Suite	✓	✓	–	–	–	N/A	✓

	V_8	V_9	V_{10}	V_{11}	V_{12}	Coverage
Acunetix	✓	✓	✓	–	–	5/11 (45.5%)
HP WebInspect	✓	✓	✓	–	–	5/11 (45.5%)
Burp Suite	✓	✓	✓	–	–	6/11 (54.5%)

Table 8.12: Vulnerability detection results for Web Application Vulnerability Scanners. For each vulnerability scanner, the vulnerabilities that were discovered are indicated with a checkmark.

required to obtain a security seal are far from rigorous.

8.4.4 Attacks

In Section 8.4.3, we showed that seal providers perform very poorly when it comes to the detection of vulnerabilities on the websites that they certify. One, however, could still argue that such products operate in a "best effort" manner and that, despite our findings, they still provide some tangible security benefits. In this section, we show that, paradoxically, third-party security seals can assist attackers in identifying vulnerable targets, and even provide them with the exact vulnerability. In addition to attacks against seal-using websites, we also show how an attacker can, in some cases, use seal providers to attack non-seal websites and how an interested party (attacker, or sketchy webmaster of a vulnerable website) can trivially evade detection by a seal provider.

Security Seal as an Oracle

Being able to accurately determine whether a website contains a vulnerability is incredibly useful for attackers since it allows them to focus their attention on websites that are likely to provide some yield.

The way security seals are currently displayed on websites, enables an adversary to pick the easiest prey from a herd of seal-utilizing websites. More precisely, security seals, which are generally hosted on the servers of the seal providers, should only be visible when a website is found to be secure. This means that if a vulnerability is found on a website, and the webmaster fails to mitigate it within the allotted grace period, the seal will stop showing. During our vulnerable webshop experiment, as discussed in Section 8.4.3, we discovered that when a seal provider wanted the seal to stop showing, they would either make the image transparent or provide an image with 1x1 dimensions.

Because of the difference in image size or content, it is possible to determine the security status of seal-using websites in an automated way. Thus, an attacker could set up a crawler to daily visit seal-using websites and be alerted whenever the image of a seal changes. To evaluate the feasibility of this attack scenario, we conducted the following experiment: for a two-month period, we visited the set of 8,302 seal-utilizing websites on a daily basis and extracted the security seal that was shown on each website. In addition, we also stored the webpage of the security seal provider that results when a user clicks on the image of each seal.

Table 8.13 shows the results of this two-month-long experiment. From all the seal-using websites, we discovered that 333 websites (Column 2 of Table 8.13) were given an invisible security seal for at least one day of our experiment. That is, in 333 cases, a website's seal transitioned from a showing seal to an invisible one, or vice versa. This indicates that either a

Seal Provider	# Sites w/ changing seals	# Sites w/ blank seals	# Sites most likely vulnerable
McAfee SECURE	260	64	-
HackerProof	3	11	-
WebsiteProtection	19	52	46
Qualys	27	15	20
Trust-Guard	5	23	-
BeyondSecurity	19	24	38
ScanVerify	0	1	-

Table 8.13: The number of websites whose security seals never appeared, or disappeared and reappeared during our two-month-long experiment.

website went from secure to vulnerable for at least one day, or was vulnerable for a series of days and went back to secure when the webmaster mitigated the discovered vulnerabilities. Related to this, for 189 websites in our dataset (Column 3 of Table 8.13), the seals were constantly invisible for the entire monitored period.[4]

This could either be due to an expired contract between the seal-using website and the seal provider, or due to a website being constantly vulnerable. In any case, from an attacker's point of view, an invisible seal should provide more than enough motive to start attacking a website.

Apart from the side-channel of seals appearing and disappearing, we discovered that for three seal providers, the combination of seal appearance and status page was different when a website was no longer a client of the seal provider, to when a website was vulnerable. For instance, for one seal provider, the seal would remain intact but the status page was indicating that the date of the last successful scan was prior to the current date. As such, for these three cases, shown in the last column of Table 8.13, we can relatively safely conclude that the seal-using website was vulnerable during our monitored period.

While a pointer towards a vulnerable website is already of great help to attackers, a disappearing seal does not pinpoint the exact vulnerability necessary to exploit a website. In the context of security seals, however, attackers can, in some cases, elicit the exact vulnerability out of a seal provider.

In our evaluation of the thoroughness of the security checks done by seal providers, we noticed that for each scan a very similar set of requests were made. These requests checked, among others, for the presence of certain files, whether a parameter was reflected without encoding, or whether a certain SQL statement would be executed. By setting up a website and purchasing a security seal (or getting a free trial), the attacker can collect the series of requests and replay them to the vulnerable website, thus discovering the exact vulnerability that caused the victim's seal to disappear. In the cases where the scan of a seal provider is dependent on the web application discovered, the attacker could set up the same web application as his victim and thus collect relevant, probing requests. An attacker could even try to replay the requests that he receives on his server directly on the victim website and extract discovered vulnerabilities in a MitM fashion. Note that once an attacker collects a trace of attack requests, he can reuse them an "infinite" number of times against vulnerable websites of that seal provider.

It could be argued that if attackers had in their possession a tool that could check the security of websites, they could run that tool against an arbitrarily large number of websites. However, we experienced that the security scanners often made a substantial number of requests, in one case up to 180,000, to the probed web server. Running such a scan on a large number of websites would require access to a considerable amount of resources, something the average wrongdoer

[4]Note that the sum of these seals in Table 8.13 is 190, since one website was including seals from two different providers.

may not have. As such, it would not come as a surprise if an attacker would prioritize attacking a website known to contain a specific exploitable vulnerability. Lastly, it is worth reminding the reader that, as shown in Section 8.4.2, more than a third of all seal-utilizing websites are e-shops, thus holding the promise of personal and financial information that are not typically present on an average website.

Cloaking

When an attacker compromises a website, it is in his best interest to keep this hidden from the website owner. In case the adversary uses the website to host malware and infect the site's visitors, the task of hiding the compromise becomes more difficult. Not only could a change in the website alarm the website administrators, but regular crawls by various search engines will also look for the presence of malware, in order to protect the users of these search engines.

To prevent detection, attackers make use of *cloaking*, where the malware distinguishes between visits of crawlers and human users and only exposes itself to the latter. Among others, attackers can use implementation-specific JavaScript code to distinguish between JavaScript engines (and thus their housing browsers), as well as cloaking at an IP-address level [168].

For some seal providers, we noticed that their scanners do not execute JavaScript. As such, an attacker can simply hide the presence of malware by testing for JavaScript support. Alternatively, attackers can, unfortunately, always resort to cloaking at a network level. During the tests described in Section 8.4.3 we witnessed that the scanning requests of seal providers were always originating from the same IP range, often a block that is registered to the seal provider. It would thus be straightforward for an attacker to only expose his malware in case a request does not originate from an IP address related to a seal-provider. This way, an attacker could easily compromise a seal-utilizing website, while the website owner would remain under the impression the website was still secure as a consequence of the daily or weekly successful seal scans. Note that this detection can be done in a straightforward manner, and is already used by attackers for conducting blackhat SEO [112].

Next to attackers, website owners could also be interested in hiding weaknesses from a seal provider. In case a seal provider finds vulnerabilities on a website, it may take a considerable amount of time and resources for the website administrator to mitigate them. If this does not happen within the grace period provided by the seal provider, the security seal – a product the webmaster paid for – will disappear. Hence, in some cases, it could prove very useful for a webmaster, if the security provider is not able to find any vulnerabilities. As such, a webmaster may be tempted to also employ a cloaking technique to deceive seal providers. In our vulnerable webshop experiment, we managed to circumvent the detection of vulnerabilities by rerouting all traffic originating from seal providers to a static web page. For all seal providers, this could be done by merely adding two lines to our webserver's configuration file. This, unfortunately, requires much less effort than continuously mitigating vulnerabilities, and could thus be employed by sketchy website owners who just want to convince their customers their website is secure.

Abusing security seal services

In previous subsections, we showed how seal providers can be abused to attack the websites that they certify as secure. In this section, we describe how they can also be weaponized against users, as well as against third-party websites.

Phishing In an attempt to acquire sensitive credentials for websites, adversaries often create phishing pages which typically resemble the original website which the phisher is targeting. To trick a user in entering her credentials, attackers can try a series of techniques to make the victim believe that she is on the legitimate website. For instance, by registering a domain that looks similar to the original domain, attackers can often convince users they are on the genuine

website. Additionally, when the original website contains a security seal, the attacker could replicate this seal on his phishing webpage to increase his credibility. Moreover, if the claims from seal providers are correct, i.e., that the appearance of a seal leads to an increase of trust in the webpage, the victim will feel safe on the phishing page.

To counter this type of attack, seal providers should only allow a seal to be included from the authentic website which they certify. In our evaluation, we found that two seal providers would not display a security seal in case the `Referer` header did not match the sealed website. The seal, however, would appear if the referrer header of a user's HTTP request was absent, which can be trivially achieved by the use of the appropriate value for the `meta referrer` HTML tag. As a result, an attacker can currently include a security seal on his phishing page from *all* ten seal providers that we evaluated. Note that these seals are fully functional in the sense that the potential phishing victims can click on them and be assured, by a page hosted the seal-provider's domain, that the seal is legitimate (not just a copy) and that the seal-bearing website is secure. If a user is already considering a phishing page enough to click on the security seal, it is unlikely that he will spot the fact that the domain mentioned in the seal provider's page is different than the one of the phishing page.

Attacking third-party websites Since seal providers search for vulnerabilities on websites, it is obvious that only the webmaster of a specific website should be able to request a vulnerability scan for that website. This is especially important since the attacks will not be launched directly by the attacker, thus making him harder to trace by the victim website at a later time.

Even though seal providers do attempt to verify ownership of a website, we found that often their methods are bypassable. For instance, several seal providers performed owner verification through the upload of a specific file on the website that requested a seal. The uploaded file should have a specific randomly-generated filename and contain a randomly-generated string. For three seal providers, we discovered that the contents of the uploaded file did not have to be an exact match as the ones provided. Consequently, if an attacker would be able to partially control the content on a URL containing the requested filename, he would be able bypass the ownership verification and get a security report for that domain. While this may seem unlikely, for several websites this can be easily achieved. On Twitter, for example, users are appointed their own URL containing their username, so a user named `foobar` is reachable at `http://twitter.com/foobar`. As such, an attacker can register an account with a username equal to the filename requested by the seal provider, and include the contents of the file in his first or last name. The seal provider will then successfully discover the "uploaded file" and proceed to perform a security scan and report the discovered issues to the attacker.

8.4.5 Discussion

We have showed that, given the current state-of-practice, third-party security seals are not only of limited value, but that they can also be used to attack seal-using websites, as well as their users, and third-party websites. In this section, we briefly describe the ways in which seal providers can substantially better their services.

In terms of vulnerability discovery, we witnessed that the vulnerability scanners of some seal providers were not executing client-side JavaScript. Given the ubiquitousness of JavaScript, we argue that JavaScript support is a necessary feature of any modern vulnerability scanner. For the seal providers that did find some vulnerabilities, we believe that their tools can be bettered if they are tried against web applications with known vulnerabilities so that the developers can quantify the coverage of their tools and prioritize the development of support for the missing functionality. For the seal providers that found no vulnerabilities whatsoever, it is clear that either their tools are fully ineffective, or that they are trying to combine incompatible technologies, e.g., searching for web application vulnerabilities using a network-level scanner.

For the problem of cloaking, a straightforward solution is to employ, from time to time, the use of VPN or cloud services, to ensure that the IP addresses of the scanners are not publicly

traceable back to the seal provider. The resulting pages can then be compared to pages retrieved from the seal provider's usual IP block using a wide range of techniques, such as text shingles [46], screenshot comparison, or comparison of the HTML structure between the two pages. Pages with large differences can be manually inspected by a human analyst who can then reach out to the webmaster of the seal-using website. To avoid being abused by phishing pages, seal providers can stop showing the security seal when the referrer header is absent, or does not match the certified website. All browsers allow, by default, the sending of the referrer header, thus the change will not affect the majority of web users who do not alter the default configuration of their browsing software.

The problem of seals being used as a vulnerability oracle is, unfortunately, not straightforward to solve. When a seal provider hides a seal as a reaction to a discovered vulnerability, this event is detectable and abusable by an attacker, as discussed in Section 8.4.4. Alternatively, if a seal provider does not hide, or in some way alter, the presence of a seal when a vulnerability is detected, then the certification power of a seal is compromised because any website can acquire it regardless of the presence or absence of vulnerabilities. Thus, given the current architecture of security seals, the honesty of a seal provider, and the security of a seal-using website seem to be contradictory goals.

The only solution to this conundrum appears to be a vigilant webmaster. If a webmaster is able to fix a discovered vulnerability within the grace period allotted by the seal provider, then the vulnerable website can avoid detection by attackers, while the seal provider can maintain its certification honesty. As such, the companies that do not currently provide a grace period – possibly thinking that this results to higher security standards – are actually doing a disservice to their clients.

8.4.6 Related Work

While there has been some evidence of the improper certification done by security seal providers, this evidence is either anecdotal or gathered in an ad-hoc manner for a handful of websites [107, 204]. To the best of our knowledge, our report is the first to systematically evaluate the certification of popular third-party security seals, at a large scale and from a security point of view. In this section, we review prior work on all aspects of third-party seals and the effectiveness of automated vulnerability scanners.

Third-party Seals Third-party seals originally received attention from the user-interface and economics community, where researchers tried to identify whether a seal is recognizable and whether it leads to an increase of trust and thus increased sales for the seal-bearing website. In 2002, Head and Hassanein [93] discovered that the awareness and influence of third-party seals was low and calling for more research on the placement and appearance of seals that would increase their awareness level. Belanger et al. [37] studied the role of various factors in how trustworthy a website appears to consumers, including third-party security seals. The authors discovered that the users valued security features (such as the presence of encryption) more than security seals. Kimery et al. discover that while a consumer's attitude in online merchants is positively correlated with the consumer's trust of those merchants, the presence of third-party seals does not affect the consumer's trust [117].

Interestingly, not all researchers have reached the same conclusions. Houston et al. [105] survey 106 students and discovered a positive correlation between the presence of a seal and the perceived quality of a product which is in turn correlated with a consumer's willingness to buy. Hu et al. [106] empirically examined the effects of various seals and, contrary to the aforementioned research, concluded that most seals that deal with trust *do* in fact increase a consumer's intent to purchase. Later research by Kim et al. [116] showed that seals have a positive effect of reducing security-related concerns of consumers, but only *after* the consumers are educated about security and privacy threats, as well as the role of third-party security seals.

The seal providers themselves rely mostly on testimonials by existing clients in order to convince businesses to adopt third-party seals. The majority of the testimonials, as well as much of the seal-provider's own advertising, revolve more around the increased sales and monetary gains resulting from the adoption of a security seal, rather than the number of vulnerabilities discovered.

Edelman [72] approached third-party seals from a different angle. Instead of investigating the effect of seals on consumers, he investigated the trustworthiness of sites that employ third-party trust seals. Trust seals are different from the security seals that we investigated in this section, in that they certify a merchant's trustworthiness, instead of their security. These trust seals are typically focusing on the privacy policy of online merchants and attempt to ensure that the merchant does not abuse the details collected by their customers. Edelman argued that a trustworthy company does not need an external entity to certify its trustworthiness. By using "site safety" information provided by SiteAdvisor, he found that websites using these seals are more likely to be untrustworthy than websites that do not use trust seals.

James and MacDougall investigated the security seals of McAfee and Trust-Guard and also reached the conclusion that the vanishing seals could be used as a security oracle [111]. The authors implemented a tool called Oizys which would automatically discover missing seals and report the potentially vulnerable websites. Our attack described in Section 8.4.4 expands upon their work by, not only locating vulnerable websites, but also discovering the exact vulnerabilities of each website via the use of a malicious proxy.

Effectiveness of Automated Vulnerability Scanners A core component of a third-party security seal vendor is the automated vulnerability scanner that dictates whether a website should be "awarded" the seal or not.

Doupé et al. conducted the most recent survey of the thoroughness of black-box automated vulnerability scanners, by evaluating eleven different scanners against a known vulnerable web application [66]. The authors discovered that many of the evaluated scanners could not cope with modern web application technologies, such as the presence of JavaScript, and thus could not fully explore websites, much less discover all known vulnerabilities in a test web application.

One difference between security seals and generic web application scanners that is worth mentioning, is the latter's higher tolerance for false positives. In a common penetration test, the tester can go through a relatively large number of false positives without any adverse effects for the tested service. Contrastingly, security seal vendors do not have this luxury, as false positives that stop the seal from appearing on their clients websites will not be tolerated by the clients who pay for the certification of their websites. As such, it is possible that the scanners of third-party seal providers choose to err on the safe side in order to lower or altogether avoid false positives which, almost unavoidably, will result in less true positives.

8.4.7 Conclusion

Providers of security seals claim that websites that make use of their services will appear more trustworthy to the eyes of consumers and will thus have an increase in their sales. In this report, we put the security guarantees of seal providers, i.e., the guarantees that indirectly influence a consumer's feelings of trust, to the test. Through a series of automatic and manual experiments, we discovered that third-party security seals are severely lacking in their thoroughness and coverage of vulnerabilities. We uncovered multiple rudimentary vulnerabilities in websites that were certified to be secure and showed that websites that use third-party security seals do not follow security best practices any better than websites that do not use seals. In addition, we proposed a novel attack where seals can be used as vulnerability oracles and describe how an attacker can abuse seal providers to discover the exact exploit for any given vulnerable seal-using website.

Overall, our findings show that current state-of-practice of third-party security seals is far from ideal. While we propose steps that seal providers can take that will substantially increase

the accuracy and effectiveness of their security certification, the issue of inadvertently creating a vulnerability oracle seems to be central to the current architecture of security seals and appears to not have a technical solution which does not sacrifice, either the honesty of a seal provider, or the security of the certified website.

Chapter 9

Observable Gaps between the State-of-the-Art and the State-of-the-Practice (DS.3)

In the Web-platform security guide (STREWS deliverable D1.1), the consortium systematically assessed the security of the current Web ecosystem, based on a set of relevant assets within the web application model, covering the infrastructure, the web applications and the user-centered assets. For each asset, the relevant threats have been identified that an attacker can apply to compromise an asset. The Web-platform security guide aimed to cover the body of work currently known in the academic world, the standardization bodies, and the developer-centered Web security communities.

9.1 Overall complexity of the web landscape

Figure 9.1 shows a combined view of the results of the Web-platform security guide, showing the assets, the threats that can lead to their compromise, and the concrete attacks embodying these threats. The combined tree aggregates a lot of information and relations, but also clearly indicates the complexity of the Web security landscape. Web security is not limited to securing individual assets, but requires a broad, integrated approach.

Table 9.1 summarizes the assets and attacks investigated in the security guide, and illustrates how the various attacks impact the web assets. The table can be interpreted in two ways:

- On the one hand, Table 9.1 illustrates the impact of a particular Web attack on the various assets of the Web platform. For instance, a successful *injection attack* can potentially impact 8 of 10 assets of the Web platform.
- On the other hand, Table 9.1 enumerates the list of attacks that need to be mitigated in a particular Web application in order to protect an asset. For instance, in order to fully protect *application transactions*, at least 17 attacks need to be mitigated.

The overview of the Web security threat landscape clearly illustrates the complexity of the Web ecosystem. To improve the end-to-end security, it is necessary to raise the bar on several (if not all) topics in parallel.

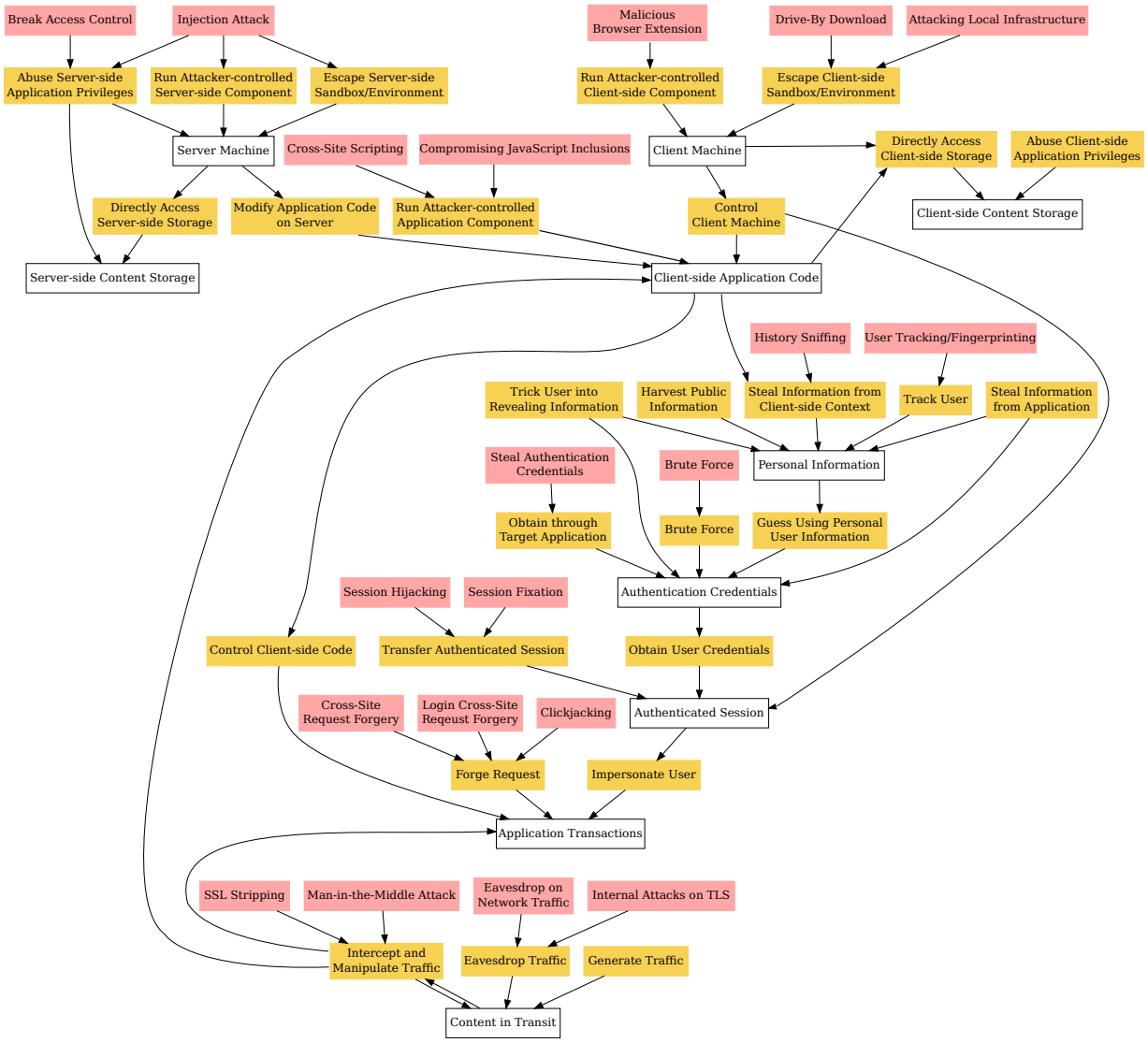


Figure 9.1: A combined view aggregating the results from the security assessment, showing assets (white), their associated threats (yellow) and the concrete attacks embodying a specific threat. The combined view not only shows the threats for each individual asset, but also identifies interesting relations and dependencies, allowing for a potential escalation of an attack.

9.2 Limited adoption of the best practices

There exists a remarkable mismatch between state-of-the-art mitigation techniques and best practices being available for almost all vulnerabilities, and we measured only a limited adoption of the best practices in the state-of-practice.

The question remains as to how web site owners can be incentivized to actually deploy best practices on their sites? Similarly, to track the adoption rate over time, it is important to have good metrics and measurements in place to be able to assess the state-of-practice of the Web ecosystem.

In the next paragraphs, we investigate how well the best practices are adopted in websites worldwide.

	Server Machine	Client Machine	Server-side Content Storage	Client-side Content Storage	Content in Transit	Client-side Application Code	Authenticated Session	Application Transactions	Authentication Credentials	Personal Information
Session Hijacking							⋆			
Session Fixation							⋆			
Brute Force									⋆	
Stealing Authentication Credentials									⋆	
Cross-site Request Forgery								⋆		
Login Cross-site Request Forgery								⋆		
Clickjacking								⋆		
Eavesdrop on Network Traffic				⋆	⋆	⋆	⋆	⋆	⋆	⋆
SSL Stripping				⋆	⋆	⋆	⋆	⋆	⋆	⋆
Man-in-the-Middle Attack				⋆	⋆	⋆	⋆	⋆	⋆	⋆
Internal Attacks on TLS				⋆	⋆	⋆	⋆	⋆	⋆	⋆
Cross-site Scripting				⋆		⋆	⋆	⋆	⋆	⋆
Compromising JavaScript Inclusions				⋆		⋆	⋆	⋆	⋆	⋆
Malicious Browser Extensions		⋆		⋆		⋆	⋆	⋆	⋆	⋆
Drive-By Download		⋆		⋆		⋆	⋆	⋆	⋆	⋆
Attacking Local Infrastructure		⋆		⋆		⋆	⋆	⋆	⋆	⋆
Injection Attacks	⋆		⋆	⋆		⋆	⋆	⋆	⋆	⋆
Break Access Control	⋆		⋆	⋆		⋆	⋆	⋆	⋆	⋆
User Tracking/Fingerprinting							⋆	⋆	⋆	⋆
History Sniffing							⋆	⋆	⋆	⋆

Table 9.1: Overview of the Web security threat landscape: mapping assets from part II to attacks from part III.

9.2.1 Impersonating users

As mentioned in the Web-platform security guide, the best practice for preventing web session attacks is:

- to use strong, random session identifiers [184]
- to renew session identifiers on every privilege change
- to deploy the application over HTTPS
- to use the *HttpOnly* and *Secure* attribute for all cookies not needed by JavaScript, especially the cookies containing a session identifier.

Unfortunately, many sites still use unprotected cookies to store session identifiers, leaving users vulnerable to session hijacking attacks. On the bright side, the adoption of the *HttpOnly* and *Secure* attributes is gaining ground, starting to be turned on by default [26].

Our July 2014 survey of the Alexa top 10,000 domains shows that more than half of the domains use the HttpOnly attribute (5,465 domains in total), and 1,419 domains use the Secure attribute, which is a significant increase compared to a study from 2010 [49].

Similarly, the large scale analysis of the European Web (Section 8.1) indicates that 33.51% of the websites deploy HttpOnly cookies and 5.33% deployes secure cookies, whereas both technologies are already readily available since 2007.

9.2.2 Forging requests

Best practices to protect against CSRF attacks are twofold, combining token-based approaches with selective unforgeable user involvement. For instance, web developers should deploy token-based approaches to protect state-changing operations. In addition, it is recommended that user involvement (e.g. reauthentication) should be required for truly sensitive operations, especially when they have direct financial consequences, or can lead to the compromise of a user account.

CSRF is prevalent in modern web sites, and is ranked in both the OWASP Top 10 project [220], listing the 10 most critical web application security risks, and the CWE/SANS Top 25 Most Dangerous Programming Errors [134]. Both small and large scale projects are affected, with for example CSRF vulnerabilities in online banking systems [223], Gmail [98] and eBay [120].

Similarly, the large scale analysis of the European Web (Section 8.1) indicates that only 16.70% of the websites deploy CSRF tokens.

9.2.3 Attacking through the Network

While TLS effectively mitigates network attacks, it is not yet widely deployed, although adoption is growing. Additionally, older and less secure versions of TLS that are deployed, are rarely upgraded to the latest version, leaving a trail of inadequate legacy implementations across the Web. For instance, the w3techs monitoring site[1] reports that approximately 36% of the top 10-million web sites are using TLS with certificates issued by a recognized CA.

SSL stripping attacks *SSLstrip* [133] intercept the HTTP traffic from the client, and impersonates the client to initiate HTTPS communication with the server, serving the page's contents over HTTP to the victim's browser.

Best practices to prevent downgrading attacks on HTTPS connections are twofold: ensure that the web site is only reachable over HTTPS, and deploy a long-lived HSTS policy, instructing browsers to connect over HTTPS.

Although the HTTP Strict-Transport Security (HSTS) technology already exists for over three years in Firefox and Chrome, the large scale analysis of the European Web (Section 8.1) indicates that only 0.50% of the websites deploy HSTS headers.

9.2.4 Controlling the Client-side Context

Injection vulnerabilities leading to XSS attacks are prevalent in both new and legacy web applications, and are highly ranked in both the OWASP Top Ten [220] and the CWE/SANS Most dangerous programming errors [134].

The best defense against XSS attacks is to apply proper input and output sanitization. Sanitization has to be context-sensitive, so one should use either sanitization libraries that automatically determine the correct context, or use the appropriate sanitization function for the output context at hand. If possible, the mitigation techniques should be complemented with a tight Content Security Policy on to prevent dangerous inline content and to limit the sources of external content.

Although the problem is well-understood, the results of the large-scale DOM-based XSS empirical study (Section 8.3) unfortunately found 6,167 XSS vulnerabilities in 480 different domains (which accounts for 9.6% of the 5000 most frequented Web sites and their sub-domains).

Similarly, the large scale analysis of the European Web (Section 8.1) indicates that only 0.06% of the websites deploy Content-Security policies, although being available in Firefox and

[1] `http://w3techs.com/technologies/overview/ssl_certificate/all`

Chrome for three years. A possible reason for the low adoption is the limitations CSP enforces (such as no inline scripts or no use of *eval*()) which makes the technology hard to apply on legacy websites. Similarly, the large scale analysis of the European Web (Section 8.1) indicates that only 0.06% of the websites deploy Content-Security policies, although being available in Firefox and Chrome for three years. A possible reason for the low adoption is the limitations CSP enforces (such as no inline scripts or no use of *eval*()) which makes the technology hard to apply on legacy websites.

Many websites suffer from *mixed content*, violating the security guarantees of HTTPS pages by including script, images or other resources over HTTP. A recent study shows that 26% of the TLS-protected Alexa Top 100,000 Web sites included JavaScript over HTTP [50].

9.2.5 Attacking the Client-side Infrastructure

A best practice for any web user is to limit the number of extensions and plugins to the minimum, and uninstall or disable those that are not or infrequently needed.

A recent study of augmented browsing script markets [199] however illustrates the popularity (as well as the dangers) of the Greasemonkey extension, which enables users to write scripts that arbitrarily change the content of any page, allowing them to remove unwanted features from web applications, or add additional, desired features to them.

As part of the study, not only dozens of malicious scripts have been identified waiting to be installed by Greasemonkey users, but also thousands of benign scripts were found containing vulnerabilities that could be abused by attackers. In particular, the authors were able to develop a proof-of-concept exploit against a vulnerable user script with an installation base of 1.2 million users, and elevated privileges equivalent to a "Man-in-the-browser" attack [199].

9.2.6 Directly Attacking the Web Application

The best practice to avoid SQL injection attacks is to use Prepared Statements. Prepared Statements enforce a clean separation of data and code. In a first step the query syntax is defined, in a second step the user input is handed over to the database and the query is executed. In this fashion, the attacker is not able to influence the query's syntax via the user input. One important point here, is that user input should never be used in the first step (when constructing the query syntax). Otherwise, this could re-enable the attacker to conduct an SQL injection attack.

SQL injection is ranked #1 in both the OWASP Top 10 project [220], listing the 10 most critical web application security risks, and the CWE/SANS Top 25 Most Dangerous Programming Errors [134]. Although the countermeasures are widely available, still a lot of websites are vulnerable to SQL injection. Very recently, the Drupal content management platform has to issue a *highly critical* Security Advisory [2] with respect to a SQL injection vulnerability in the project CORE.

9.2.7 Violating the User's Privacy

The best practice to prevent state-based user tracking is to keep a clean browser profile, using one of the many available extensions. Regularly cleaning the cookie store, especially cookies from unknown domains, might help as well. Unfortunately, no mitigation technique against fingerprint-based is currently known.

In [150], the authors identified several thousand websites that deploy web fingerprinting techniques, offered by one of the three investigated providers. Moreover, the authors demonstrate how over 800,000 users, who are currently utilizing user-agent-spoofing extensions to mitigate web fingerprinting, are actually more fingerprintable than users who do not attempt to hide their browser's identity [150].

[2] `https://www.drupal.org/SA-CORE-2014-005`

9.3 Trends in web security

9.3.1 Trend towards server-driven browser enforcement

Significant areas of novel web security technology (both in research and standardization) follow the same pattern: The server issues a security policy, the policy is pushed towards the client as part of the web application, and the client is responsible for enforcing the policy correctly. Well-known examples in recent specifications are CSP, X-Frame-Options, HSTS, and Certificate Pinning.

In this context, the Content Security Policy (CSP) seems to be a very promising additional layer of defense, protecting against cross-site scripting and UI redressing. In addition, CSP is an interesting enabler for more advanced security architectures, such as the architecture underneath the office document reader inside Chrome OS [211].

9.3.2 Shift from purely technical to user-centered

Web security is partially shifting from a purely technical topic to a user-centered topic. This is illustrated with the numerous phishing and social engineering attacks, and the web permission model relying more and more on decisions of the end-user. For sure, attackers will more and more target the user as the weakest link of the web infrastructure.

Moreover, UI security becomes a key factor in delivering a secure web ecosystem, especially with the rise of new web-capable devices, such as smartphones, tablets, etc. The web permissions model is extraordinarily complex, and hard to understand for the end-user. Key questions are how to involve (or not involve) users in security-related web decisions and how to communicate back security results to the user.

9.3.3 Focus on pervasive monitoring

In light of the recent revelations of Snowden [197], the security community is investigating a new attacker model, covering the pervasive monitoring and wide-scale man-in-the-middle capabilities of high-ranking agencies. Additional effects are the reinforced interest in security protocols that offer strong guarantees, such as perfect forward secrecy (PFS), as well as TLS-everywhere proposals and alternatives for the currently-deployed CA architecture. Finally, *algorithm agility* is key to guarantee the long lifetime of security-related protocols in the Web.

9.3.4 Increasing need to compartmentalize web applications

As web applications are becoming larger, and contain more third-party components (e.g., third-party JavaScript inclusions), the secure containment or sandboxing of untrusted parts of the web application becomes crucial. Current state-of-the-art containment techniques still need to mature, both in terms of policy specification as well as enforcement techniques.

9.4 Additional challenges

1. We clearly see the urge to fix some of the legacy building blocks of the web model. For instance, passwords are still the primary authentication technique on the web, and are almost always used in combination with bearer tokens (e.g., session management cookies and OAuth tokens).

   Major changes to legacy building blocks of the Web model face prohibitive deployment obstacles, as the currently-deployed legacy of web applications relies on the legacy model's properties, and the adoption of best practices is rather slow.

2. In order to advance, it is important to re-unify the specification and implementation fronts. For example, the WHATWG fork of the W3C HTML5 working group, the BLINK fork of WebKit, and the numerous disagreements on various topics (such as the use of URL/URI/IRI, the use of unicode in domain names and URIs) might confuse people or slow down the necessary security innovation.

   The various forks could potentially lead to a situation where security technology is only partially picked, or adopted at varying speeds, or even worse continue to work on potentially conflicting variations of the same technology.

3. From time to time, the moving semantic model of the web breaks the (security of) legacy applications. Interesting examples are the introduction of the application cache, or the interference of XHR/CORS with other web features, such as the traditional same-origin restrictions of XHR, or the *null* origin of an HTML sandbox [62].

4. Updates to the foundations on which the Web ecosystem is built, often impact certain aspects within the Web ecosystem, regardless of all attempts of achieving a cleanly-separated, layered approach. Recent updates requiring additional attention are the potential interference of the ongoing deployment of DNSSEC, or the IPv6 rollout, for example in code or policies that explicitly deal with bare IP addresses.

Chapter 10

Interactive Survey (DS.4)

STREWS first created a questionnaire with the questions noted down in D3.1, the Intermediate Roadmap for European Web Security Research Landscape. The 48 sophisticated questions were validated and put on W3C's Web based balloting system (WBS)[1]. The survey was widely promoted with W3C homepage news, STREWS website and twitter. Unfortunately, only very few people filled the sophisticated survey. We did fall back to a very simple survey with 5 questions on a constrained time scale. We obtained many more responses.

10.1 The complex Survey

The complex survey was presented in D3.1, the Intermediate Roadmap for European Web Security Research Landscape. This complex survey featured 48 questions and was constructed using W3C's Web-Based Straw-poll and balloting system (WBS). In order to avoid duplication and multiple answers, people had to identify themselves with an email address. A lot of promotion was done, but the complexity of the questionnaire finally seemed to frighten people. The survey only received a few answers. It is nevertheless worthwhile to report some very interesting results from those few answers. Most respondents use content management system for their web presence. Most of them still use cookies, some use headers. Most of the layout is done using Cascading Style Sheets and only a few people use javascript. If they do so, they use javascript locally (on their server). Everybody uses XMLHTTPRequest, mostly together with a SQL database. This seems to be the bread and butter setup. Some people admitted that while using advertisement, they don't know what data the ad network collects. But they had looked at the javascript code they were supposed to put on their site to make the ads work. Most of the respondents created their web presence by using multiple other sources to pull content. Most where using a javacript package. It was interesting to see that most of the respondents had a local copy of the javascript library they were using. Only one respondent used TLS to load the script and only one other actually had checked the javacript library for malicious code, which indicates a high potential for vulnerabilities and XSS. Respondents said they use very widespread javascript libraries and thus trust that there would be noise if something was wrong with the package. Many just use a javascript package they find on the network. If HTML5 gets more widespread, this can be a new very proliferate vector for attacking sites and their auditorium. Most respondents still identify people by a combination of login/password. Half of the respondents use outh and authentication with ID-accounts from larger providers or social networks. TLS is already enabled on half of the sites but they all use one of the well known CAs. DANE and DNSSEC seem still far away. Mitigation techniques like CSP, X-Frames and HSTS are more widespread than expected. The seemingly more professional respondents also had a software life cycle concept. Most of the respondents believed that better browser interfaces

[1] https://www.w3.org/2002/09/wbs/1/strews-roadmap/

would improve the fight against phishing. But when asked about the security interface of their smartphone, the responses were spread evenly between not enough, sufficient and not enough. Many of the respondents use geolocation in their service.

This was interesting feedback, but this has to be taken with care as the sample was very small. And because the sample was so small, STREWS created a simplified quick Survey.

10.2 The simplified Survey: Execution and Results

The simplified Survey was done using Surveymonkey. This time it did not require any identification as identification was optional. No email address was needed and there was no check done on duplications. The URI of the Survey was distributed via the STREWS Advisory Group, the relevant IETF mailing-lists, the relevant W3C mailing-lists and social media. As the time for the simplified Survey was very short, there was no second round of notification.

The simplified Survey attracted 34 answers, most of them within 24 hours after the parallel announcement in the multiple channels mentioned. As STREWS has a community and was reaching out to the community, the responses were mostly from people of the security community. Nearly all respondents have a background in software engineering, most of them also a background in security. The answer can be thus seen as mostly coming from experts in the field. As social media was used that reaches also other non-security communities, it can be said that those did not feel a need to answer the questions. Security seems to be a subject were people expect something but don't feel concerned enough to raise their voice. This may be due to the intrinsic complexity of the topic, or it may be that the community in social media is less concerned.

Shortening and simplifying the Survey was thus a good means to mitigate the rather narrow success of the more complex but more revealing first Survey. This way STREWS was able to validate that it had taken the right topics and was exploring in the right direction. The simplified version of the Survey had only a short time for responding. It had only 5 questions.

10.2.1 Personal information

Question	Explanation & Options
Profession	Free Text
Typical role with respect to your Web site(s)	Free Text
Optionally, in case you are participating in a European Research Project, please, leave your contact data:	• Your name • The name of the project on whose behalf you are answering this survey • The project's Web site

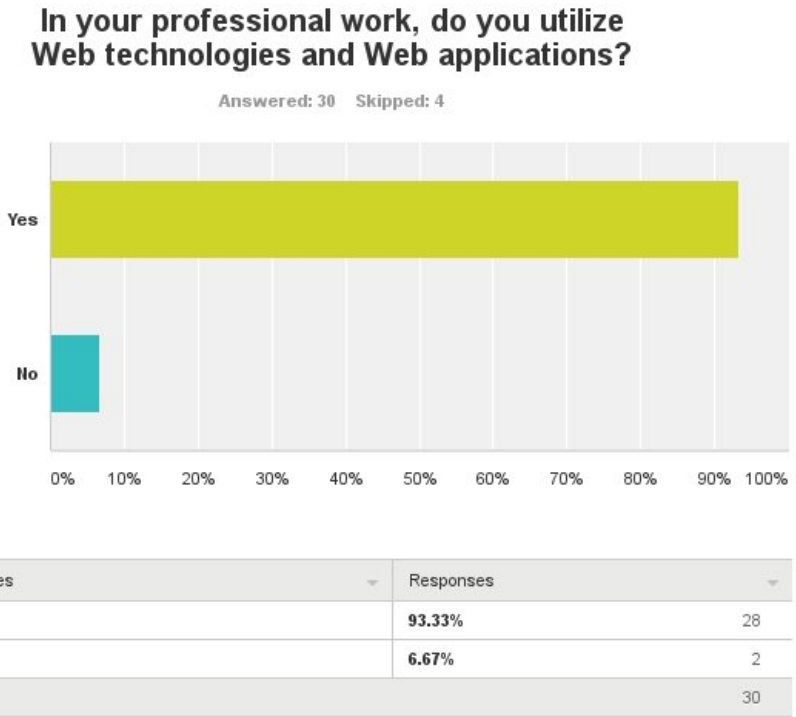
The responses can be depicted as follows:

Profession	Value in %
Software Engineer	68,75%
Security Consultant	25%
Marketing	6,25%

Nearly all respondents have a background in software engineering, most of them also a background in security. The answer can be thus seen as mostly coming from experts in the field. As social media was used that reaches also other non-security communities, it can be said that those did not feel a need to answer the questions. Security seems to be a subject were people expect something but don't feel concerned enough to raise their voice. This may be due to the intrinsic complexity of the topic, or it may be that the community in social media is less concerned.

10.2.2 Web technologies

Question	Explanation & Options
Are You using HTML, JavaScript or HTTP in Your project other than for the Project's Website?	• Yes • No



Answer Choices	Responses	
Yes	93.33%	28
No	6.67%	2
Total		30

Figure 10.1: Responses to Question 2 indicate high interest for Web technologies

Web technologies are used ubiquitously. Only 6.67% don't use them. This is a strong indication for the importance of making the Web more secure.

10.2.3 Emerging Web technologies

Question	Explanation & Options
In your professional work, do you plan to use emerging Web technologies?	• We are looking into WebRTC • We plan to move our native Applications to Web Applications • We plan to use Web Workers
Other (please, specify up to two below)	Free Text

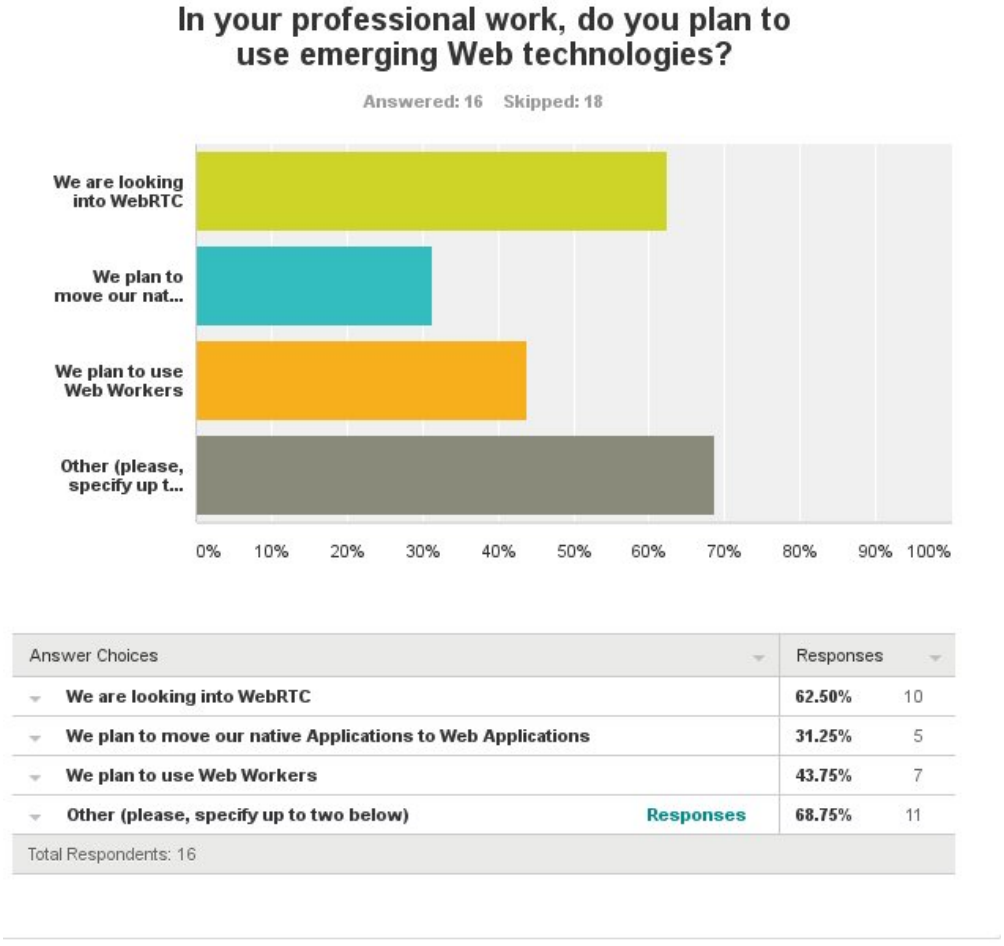


Figure 10.2: Responses to Question 3 about emerging Web technologies

The results to the question shows that STREWS was right to invest one case study into

WebRTC as the momentum behind it is clearly indicated by the answers to the third question. But the highest rank was given to the free text field. Analysing those fields yields the following ranking:

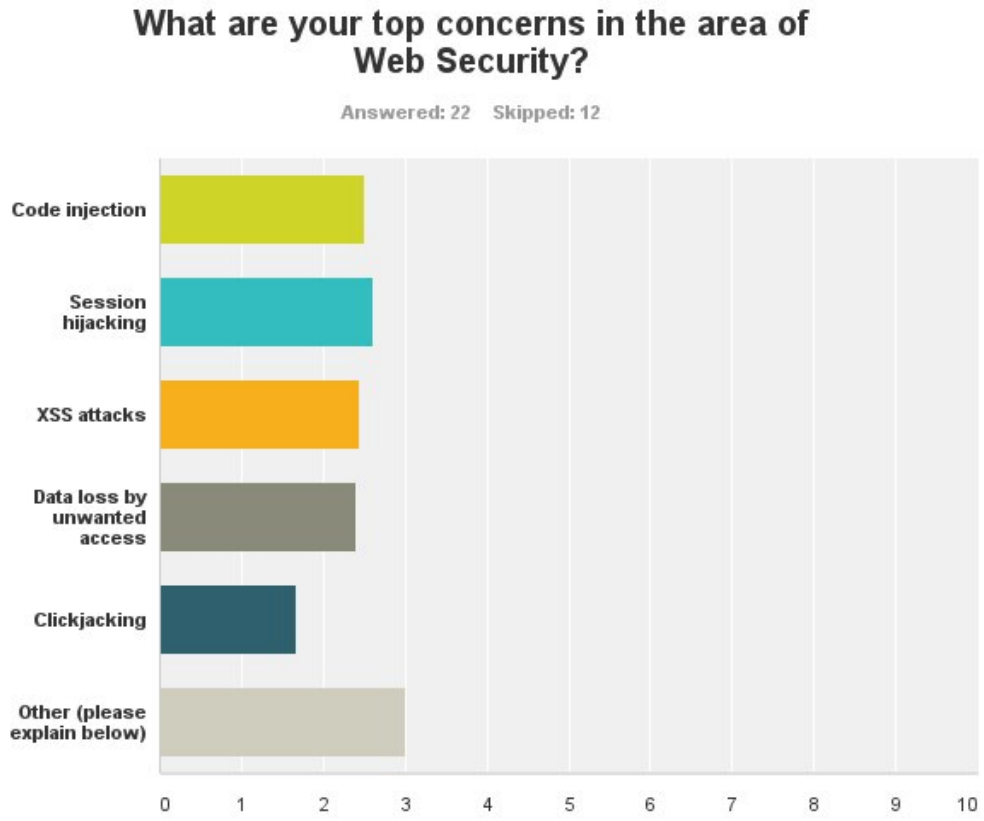
1. Content Security Policy[216]
2. Web Crypto API[185] There is already an indication that people want to use hardware tokens and TEE inside the Browser
3. Upgrade Insecure Requests[215]

But there were also critical voices alluding to the difficulties to secure the Web platform.

10.2.4 Top concerns

Question	Explanation & Options
Code injection	• low • medium • high • no opinion
Session hijacking	• low • medium • high • no opinion
XSS attacks	• low • medium • high • no opinion
Data loss by unwanted access	• low • medium • high • no opinion

Clickjacking	• low • medium • high • no opinion
Other (please explain below)	• low • medium • high • no opinion
Comment	Free Text



	low	medium	high	no opinion	Total	Weighted Average
Code injection	19.05% 4	9.52% 2	66.67% 14	4.76% 1	21	2.50
Session hijacking	4.76% 1	28.57% 6	61.90% 13	4.76% 1	21	2.60
XSS attacks	14.29% 3	23.81% 5	57.14% 12	4.76% 1	21	2.45
Data loss by unwanted access	13.64% 3	31.82% 7	54.55% 12	0.00% 0	22	2.41
Clickjacking	52.38% 11	9.52% 2	23.81% 5	14.29% 3	21	1.67
Other (please explain below)	0.00% 0	0.00% 0	75.00% 6	25.00% 2	8	3.00

Figure 10.3: Responses to Question 4 on top concerns of Web Security

The STREWS assumptions in aligning the questions in the questionnaire were shared by the respondents. Code injection was seen as the highest threat with the other possible attacks following. In the comment field, respondents had the opportunity to mark additional concerns. Further remarks included the following:

- XSS attacks are really issues with web browsers: there just shouldn't be a problem.
- Complexity. Inability to rely on security of underlying tools (libraries, platforms)
- Centralisation of information on the internet. We aim to distribute the information so as to reduce the Big Brother effect
- DNS, 509 issues, UI confusion

10.2.5 Biggest security challenges

Question	Explanation & Options
What do you consider to be the biggest security challenge(s) the Web (and Web applications) faces now and in the future?	Free Text

The responses to this pure text question were pretty widespread. But there were still recognisable patterns:

- Tracking and content injection by the advertisement industry was one of the biggest threats to Web security. This in turn facilitates the creation of a technology that creates ideal conditions for pervasive monitoring.
- The above goes hand in hand with the recentralisation of the Internet and the Web by large corporations controlling large parts of the technology stack was identified as a threat to security and privacy
- the complexity of the system was blamed in many responses. This included the fact that deployment of solutions on the Web is very costly, burdensome and long.
- Lack of testing and an environment that favours design over security and resilience make it harder to find the necessary momentum.
- The lack of good user interfaces for security is mentioned
- The interactivity in a system that was never designed to handle it

10.3 Survey Analysis

Despite the rather low sample, both surveys bring new aspects to be considered on the one hand. But they also confirm to a large extend the assumptions STREWS made based on other data sources and via the constant communication within IETF, W3C, research and the Commission initiatives around Cybersecurity. The main new aspect is certainly the extend to which tracking is seen as an issue and even interfering with security. A confirmation is that STREWS did well in taking up emerging Web technologies for its use cases. Further work is needed to make those emerging technologies resilient.

While the focus of this roadmap is on research, it has to be noted that the survey exposed an astonishing clarity of the security experts with respect to the re-centralisation of the Web by large corporations. This raises a political issue that can not be underestimated. Similar to the monopoly in economics, there is a risk of a very high influence by certain stakeholders that

is abused to optimise for their purposes the technology that even countries depend upon. This conflict of interest needs certainly more research to allow for an informed decision making in the future. It is essential for the deeper understanding of a Cybersecurity policy that can have success.

The Surveys both showed that the experts are conscious of the risks, that they use the security tools built for them in the Internet and Web Standardisation organisations. This means, the more we can interest security researchers for the real problems of the Web, the more we have insights about those, the faster we organise the technology transfer from research to Standardisation, the more secure the Web will be. The bottleneck will be the browser manufacturers once a security tools starts to conflict with their business model that is based on advertisement and data collection. If eGovernment wants a more secure resilient Web, there is a certain interest in investing also in client side software because the tools used by the advertisement world will not be good enough.

Chapter 11

Review of related NoE and Policy activities (DS.5)

A research roadmap must take into account the efforts that have been undertaken so far. As there is high activity in the area of research but also in the political and institutional arena, a chapter that still makes sense has to work with a reduced scope. While the ANIKETOS [11] project talks about secure service composition, it is geared more towards the web services model that is not used in the context of the mash-ups on the Open Web Platform. It is rather used in B2B scenarios that use IP-based networks to organise their extranet with some middleware that has innovative security checks in place and does runtime checks based on the WSDL Description of the service offered. Meanwhile, the mash-ups are mostly based on APIs that allow to use certain data via JavaScript handles and integrate the resulting ensemble in a - sometimes dynamic - page that is shown to the visitor of the web site. Consequently the chapter at hand only reports of activities that concern the Open Web Platform scenarios. At one notable exception: An activity that should talk about Web Security and does not is identifying a gap that inspires actions a roadmap may suggest.

11.1 Nessos

11.1.1 Overview

The Network of Excellence on Engineering Secure Future Internet Software Services and Systems (NESSoS)[6] aims at constituting and integrating a long lasting research community on engineering secure software-based services and systems. As a result of its networking activities NESSoS created a scientific Roadmap for secure software and services.

NESSoS has created a research agenda[36][180] to identify the relevant security challenges for the «*Future Internet*» that need to be addressed by research. Content and conclusions are the result of community work and a survey. NESSoS identifies the following main elements for «*Future Internet*» security research:

- Security requirements engineering
- Secure service architecture and design
- Security support in programming environments
- Service composition and adaptation
- Runtime verification and enforcement

- User-centric security
- Security management
- Autonomic security
- Quantitative aspects of security

Those are useful terms that STREWS can operate against to check whether the Web Security world is addressing those challenges and under which terms. E.g., security requirements engineering is addressed in the IETF by a procedural approach: Every RFC has to contain security considerations. While writing and implementing, the engineers will have to think about security to be able to address that mandatory section. On the other hand, the Web was always made for interoperability. Secure service and architecture design were of lesser concern. The Web Application Security Working Group in W3C tries to address some of those issues, but currently, it remains patchwork. Much remains to be done. Same counts for programming environments. As we have seen in the Web-platform security guide [64], the Open Web Platform has turned from an information distribution system into a platform for application development. But this platform still lacks a programming environment as we know it from e.g. the native mobile operating systems for their mobile applications. Research into making a security enhanced environment for Web Application development looks like a worthwhile target to gain security improvements on the Web on a large scale. Service composition and adaptation are core topics of Web Security and are addressed already by work on Content Security Policies in W3C[191] and the works on version 2. Runtime verification and enforcement is addressed by the works on subresource integrity[43]. User centric security is addressed in the W3C Workshop on Privacy and User centric Controls[180] but lacks a bigger perspective and commitment from the community to advance the state of the art. Security management, autonomic security and quantitative aspects of it are very foreign to the current thinking in Web Security. They play certainly a role to secure web servers but do not really address a narrow definition of web stack.

In large parts, the NESSoS report focuses on vertical application scenarios like eHealth and SmartGrid. Those are only of limited interest to a generic approach to Web Security. While people are looking into security requirements for eHealth, while people test if they are implementable, while people implement eHealth using the Web, none of the efforts is researching the gaps if eHealth security can be implemented on the Open Web Platform that will be used by those Services. Just defining this challenges away by assuming that a Future Internet will have the required capabilities is rather looking like a dead end. Looking deeply into the application scenarios described by NESSoS goes beyond the scope of STREWS. STREWS focuses first on making the Open Web Platform secure before checking whether very specific application scenarios are confronted with gaps of the Open Web Platform.

In section 4 of the NESSoS report[36], most of the considerations go into the software lifecycle management and its environment and periphery. It has to be noted that the considerations done under privacy, compliance, legal considerations and «*right to be forgotten*» are not dealt with under the umbrella of «*Security*» within the Web community. Privacy and compliance are rather a horizontal matter in its own right.

11.1.2 Conclusion

The NESSoS Research Roadmap is not without interest for Web Security. But it is apparent that there is a gap between security research that addresses plain application development and the research challenges in Web Security. It may be useful to explore the reasons for that gap. Because large parts of the current developments in eGovernment, eHealth and Cloud computing are based on or dependent on Web technologies. This means we may gain great insight on how to secure services, but we will not be able to implement because of missing building blocks on the Open Web Platform used by the service we want to secure.

11.2 syssec

11.2.1 Summary

syssec is a Network of Excellence in Managing Threats and Vulnerabilities in the Future Internet. In the second year of its operation it has produced a roadmap for systems security[131]. Despite an emphasis on "Systems", the Report from syssec encompasses very interesting sections that touch on Web Security. A roadmap for Web Security will have to take them into account. Areas of interest include not only where syssec call them Web vulnerabilities, but also things that have the Web as a main attacking vector like phishing and social engineering. syssec confirms that pervasive monitoring is a problem that we have to address. The STRINT Workshop has brought us much more advanced insights than those described in syssec's red book.

11.2.2 Details

syssec has an explicit mapping of Web Security, but also reports on other aspects that can be categorised into the wider field. Data breaches, social engineering and phishing are certainly part of the things Web Security must look at. Pervasive monitoring, online tracking and impersonation attacks are also widely represented on the Web. syssec delivers a lists of threats, a list of assets and a list of domains that map naturally into a matrix in the reader's head. syssec identifies 13 threats and issues the conclusions under the creative title of "Grand Challenges". The identified threats are:

- Lost Anonymity
- Software Vulnerabilities
- Social Networks
- Critical infrastructure
- Authentication and Authorisation
- Security of Mobile Devices
- Usable Security
- Botnets
- Malware
- Social Engineering and Phishing

The Web is present in most of those threats. But it is a crucial enabler for the Lost Anonymity, the Social Networks and Phishing. Improvements in the security of the Web stack will certainly lead to strong improvements in the area of Mobile Devices and Malware. In that respect, the syssec report is useful as a basis for further brainstorming about threats to the Web platform and the definition of areas for research on mitigation techniques. To exemplify, we take several of the identified threats and match them into the threats for the Web platform.

Lost Anonymity

Lost Anonymity is certainly a topic that also STREWS addresses, especially after the very successful STRINT Workshop on pervasive monitoring. syssec does not address the underlying Internet - layer that –in its current state –facilitates tracking. The syssec report is concentrating on the Web layer in a strict sense and identifies W3C's activities around Do Not Track as «state of the art». It concentrates on social networking and urges to define a measuring system to count

and rate the amount of leakage an action in social networks will generate. To attract data for the measuring, honeypots will serve to track leakage. Concentrating further on social networks, research into the question of deletion of data on the Web and in social networks is suggested.

Software Vulnerabilities

For syssec, the term software vulnerabilities obviously rimes with the usual attacking vectors like memory errors, input validation errors and race conditions. But it was a mild surprise to see cross-site request forgeries or clickjacking categorised here. As the Web is made of software, this is certainly a legal categorisation that is then labelled «*privilege confusion*». Under the term of «*insecure interaction between components*» we find the well known SQL-injection that was identified as a solved but still widely present vulnerability in typical web applications. syssec identifies the Open Web Platform and cloud computing with their wide range of APIs as fields that needs further research to prevent data exposure vulnerabilities.

Social Networks

The huge growth of social networking makes this a new fact of life in our society. Social networking provides new opportunities to create assets, reputation and added value. And of course one can use malicious acts to unduly benefit or manipulate the assets and values created. The main security risks identified by syssec include Privacy, Spam, Sybil attacks, Authentication and the modular structure allowing malicious applications on certain platforms. Interestingly enough, syssec does not identify those as research challenges, but suggests to explore the new data mining techniques used on the social graph created by social networking in order to detect unusual behaviour. Mining the social graph to detect and classify attackers and also to look at attacking dynamics over time.

Authentication and Authorisation

Authentication is used to control assets on the network. Many people over many years tried to get rid of passwords. They are still here. The syssec Roadmap is first exploring the vulnerabilities of password based systems. It is remarkable that the report accurately describes the risk of authentication tokens and abilities in a few words by describing the loss of identity in social networks. This is interesting because it shows how much people value their reputation in social networks and also demonstrates that this reputation is as vulnerable as the systems the social network is running on. OpenID and Oauth are described as state of the art. Some people may differ here, as there is still a long way to go before those can be seen as reliable and deployed. The syssec report suggests to research on richer authentication beyond passwords. It also addresses the vulnerabilities created by massive service federation between the biggest identity providers from search and social networking.

Usable Security

The syssec authors recognise that usability of security is a very old problem that hasn't been solved and now faces additional challenges. They call it «*complexity*» and mean all the sensors and different devices that talk to each other and are synchronised (or not), the multitude of accounts that everybody has on the large varieties of web sites we use. The report erects principles for Security UI: Simplicity, Transparency and Restrictiveness. While the first two are more or less self-explanatory, the third is the attention given to only use security if needed to avoid restricting the users in what they try to achieve. According to syssec, it comes down to the observation that a little more complexity is acceptable for a fair offering in value.

The research agenda they suggest sounds compelling:

- Usability guidelines for security research. It is true that we need a basic collection of do's and dont's for user interaction

- Usability field study. Here they suggest testing the usability of existing solutions. Something that certainly would also benefit the Web.
- Collateral Feasibility Considerations to incorporate usability decisions in the development process right from the beginning.

11.3 Building International Cooperation for Trustworthy ICT (BIC)

The Web is global by nature. The BIC-Project (Building International Cooperation for Trustworthy ICT: Security, Privacy and Trust in Global Networks & Services) had a global perspective in its name. Looking at the final recommendations[172], most of it is country reports and some indigestible mix of philosophy, policy analysis and false assumptions about the Internet and the Web. STREWS will not use this report as an input.

11.4 SWEPT

SWEPT[14] proposes a security solution that incorporates different security mechanisms and tools for automatically mitigating web site attacks, and thereby maximising the security posture of websites with a minimum of intervention from web site owners and administrators. The project has just started. SWEPT will make available for website administrators and owners an innovative set of measures for the protection of websites, to be applied at the web application level. When applied in the comprehensive way this proposal sets out, these measures, based on the security by design concept, will raise website security to new levels and with the minimum of administration and maintenance effort. STREWS participants will be on the advisory board of SWEPT and thus information from SWEPT will feed into the second iteration of the STREWS Web Security Roadmap.

Chapter 12

STREWS Workshops (DS.6)

12.1 Introduction

Workshops are a well tested and well established way to test the waters with the Web community. People meet at conferences and elsewhere and the same topic comes up over and over again. Now it is time to invite people and give them a platform for further discussion. Despite the fact that everything is called a Workshop, there are roughly three categories of Workshops:

- *Scientific Workshops:* Workshops of this type typically have a wide scope defined and call for papers. Those papers are sent to a programme committee with rather senior representatives from the scientific community that is covered by the scope. Other researchers submit papers and the programme committee proceeds with a peer review of the submitted papers. Only papers above a certain threshold of originality and quality are accepted and published in proceedings. Once a paper is accepted it will normally presented to the audience. Discussion is not the primary goal of the Workshop.

- *Standardisation Workshops:* Workshops of this type are geared towards the industry. The scope is normally very narrowly defined. In W3C those are called « *Workshop*» in the IETF they are called «*BOF*». This does not include researchers. People submit opinions and papers that address or solve the narrowly defined scope. Those papers are scrutinised by a programme committee. This time, the programme committee will publish all papers. But only a selected few papers are invited for presentation. In W3C typically, one can not get into a Workshop unless one has submitted a position paper. For BOFs this is not the same. The goal is that the presentations trigger a fruitful discussion between the participants. At the end, the commitments are collected and a decision can be made whether the narrowly defined question is ripe for joint work and standardisation. The goal of such a Workshop is work-driven discussion.

- *Hybrid workshops :* In the middle between both worlds, there is a lot of grey area. The Internet Standardisation Organisations typically also use Workshops to give people some orientation and to explore unsolved problems. The scope can be wider, but never as wide as the one of a scientific Workshop. In this case, a mixture of scientific papers and industry statements is submitted. The programme committee makes an interesting agenda out of all the submissions. Everybody can attend. Those workshops are preparatory acts for highly innovative steps on the Web and on the Internet. The discussion is encouraged and often leads to technology transfer from research to industry. There may be a concrete outcome, but this type of Workshop is not intended to trigger the creation of a Working Group or to trigger concrete work on a very specific item. All STREWS Workshops so far have been of this hybrid type.

12.2 STRINT

12.2.1 Motivation

The STRINT workshop was arranged, chaired and sponsored by STREWS and took as its starting point the IETF consensus that had been established to the effect that Pervasive Monitoring is an attack on the Internet [80]. As explained in the RFC 7258, the word «*attack*» is used in a very technical sense, not in the common English sense. It implies nothing about the motivation of the actor mounting the attack. The motivation for PM can range from non-targeted nation-state surveillance, to legal but privacy-unfriendly purposes by commercial enterprises, to illegal actions by criminals. The same techniques to achieve PM can be used regardless of motivation. Thus, we cannot defend against the most nefarious actors while allowing monitoring by other actors no matter how benevolent some might consider them to be, since the actions required of the attacker are indistinguishable from other attacks. This together with the experience that today's nation state techniques of pervasive monitoring are accessible to criminals tomorrow motivate the work to mitigate pervasive monitoring. A glimpse of the transition to private attackers can already be seen in the fact that authorities actually went to internet giants as they had better and higher quality data about users. In chapter 9 we have learned that most of the European Networks of excellence consider tracking of users to be a behaviour that should be subject to security considerations. Tracking is much weaker than pervasive monitoring. But most of the properties of the Internet and the Web that are used for tracking also find their utility in pervasive monitoring.

STRINT was a useful input for roadmapping for a few reasons. First, the topicality of STRINT almost guaranteed an excellent set of attendees - gathering that 100 people in a room to dicuss the future of Internet and Web security and privacy was always going to produce good input for roadmapping. Second, as is pointed out in RFC 7258 [80], attacks that are only feasible for state level actors today, will be feasible for much less capable bad actors in future. That means that plans for mitigation of the PM attack will be much more broadly relevant in future too.

12.2.2 Results

The workshop is described in more detail in STREWS Deliverable D2.2 and in the Internet Architecture Board (IAB) report [81] on the workshop, which summarises the resulting recommendations as follows:

1. Well-implemented cryptography can be effective against Pervasive Monitoring (PM) and will benefit the Internet if used more, despite its cost, which is steadily decreasing anyway.

2. Traffic analysis also needs to be considered, but is less well understood in the Internet community: relevant research and protocol mitigations such as data minimisation need to be better understood.

3. Work should continue on progressing the PM threat model draft (I-D.barnes-pervasive-problem) [30] discussed in the workshop. That Internet-draft was subsequently superseded by another [31] and is now being developed via the IAB's security and privacy programme. (And that programme was revitalised and re-constituted partly as a result of STRINT.)

4. Later, the IETF may be in a position to start to develop an update to BCP 72 [171], most likely as a new RFC enhancing that BCP and dealing with recommendations on how to mitigate PM and how to reflect that in IETF work.

5. The term "Opportunistic" has been widely used to refer to a possible mitigation strategy for PM. We need to document definition(s) for this term, as it is being used differently by different people and in different contexts. We may also be able to develop a cookbook-like

set of related protocol techniques for developers. Since the workshop, the IETF's security area has taken up this work, most recently favouring the generic term "Opportunistic Security" (OS) [68]

6. The technical community could do better in explaining the real technical downsides related to PM in terms that policy makers can understand.

7. Many User Interfaces (UI) could be better in terms of how they present security state, though this is a significantly hard problem. There may be benefits if certain dangerous choices were simply not offered anymore. But that could require significant co-ordination among competing software makers, otherwise some will be considered "broken" by users.

8. Ways to better integrate UI issues into the processes of IETF and W3C needs further discussion.

9. Examples of good software configurations that can be cut-and- paste'd for popular software, etc., can help. This is not necessarily standards work, but maybe the standards organisations can help and can work with those developing such package-specific documentation.

10. The IETF and W3C can do more so that default ("out-of-the-box") settings for protocols better protect security and privacy.

11. Captive portals, [1] (and some firewalls, too) can and should be distinguished from real man-in-the-middle attacks. This might mean establishing common conventions with makers of such middleboxes, but might also need new protocols. However, the incentives for deploying such new middlebox features might not align.

Of specific interest here, the STRINT workshop also had a subgroup who looked at research topics which is described as follows in section 5.8 of [81]:

> Despite some requests earlier in the workshop, the research break-out did not discuss clean-slate approaches. The challenge was rather that the relation between security research and standardisation needs improvement. Research on linkability is not yet well known in the IETF. But the other side of the coin needs improvement too: While doing protocol design, standardisation should indicate what specific problems are in need of more research.

The break-out then made a non exclusive list of topics that are in need of further research:

- The interaction of compression and encryption as demonstrated by the CRIME SSL/TLS vulnerability [13]
- A more proactive deprecation of algorithms based on research results.
- Mitigation for return oriented programming attacks
- How to better obfuscate so called "metadata", how to make the existence of traffic and their endpoints stealthy

[1] `https://en.wikipedia.org/wiki/Captive_portal`

12.3 Workshop on User-centric Controls

12.3.1 Motivation

The topic of usability was already present in the first STREWS Workshop [2]. Much of the security tools are already in place, many of them standardised. But they are hardly used. The Workshop on Privacy and User-centric controls[209] was arranged and organised by a cooperation of the STREWS and the Practice Project. The chair was from the Practice Project. Already in STRINT, the success of the Workshop was due to an alignment with an important question that was raised within the IETF. The Workshop on User-centric controls was using a larger discussion about security and privacy interfaces in several W3C Working Groups. STREWS was then instrumental to give this discussion a platform that allowed to find future looking actions in the very important area of security usability.

The meeting on trust and permissions for Web applications, that was held in Paris on 3-4 September 2014 has provided insights on a way for a roadmap towards a broad consensus on trust and permission handling for the Open Web Platform. There was agreement, that browsers are in a position to examine the APIs used by a given app and apply heuristics to determine signs of attempts to "finger print" the device. This could be flagged to the user as well as to potential reviewers.

While the Paris meeting explored models on how to delegate trust decisions, this Workshop explored ways to directly help the user understand what is going on. This included appropriate ways of translating complex issues involving fine grained permissions in APIs into something that users understand, including basic privacy UI features that will, on the long run, create a user experience that loops with user expectations.

12.3.2 Results

The Workshop on Privacy and User-centric Controls is described in detail in STREWS Deliverable D2.3 and in the W3C Workshop Report[209]. The recommendations of the Workshop and the further questions and actions can be summarised as follows:

Bigger Picture Questions

1. How does user control relate to controls?
2. Can we do more than minor tweaks?
3. How can we address systemic issues with point standards, for example retention and re-use
4. Is it possible to have both ads and privacy?
5. How to enable choice, shift from consent to choice? Is this a legal issue only?
6. Can we agree standardisation related to a general approach to avoid consent dialogues?
7. Is an incremental approach possible, and if so, what is the first step? Risk assessment or known need?
8. How can the community help each other; For example, what would help the Firefox OS community?
9. Can we create better regulatory linkages?

[2] STRINT see above 12.2

Awareness Questions

1. What are the confidence-inspiring metrics for security/privacy?
2. Can we standardise metrics for security/privacy that inspire confidence?
   - enumerate sources, e.g. frequency, 2nd order effects, confidentiality/security
   - define calculation
   - relate to frequencies (e.g. how often is data collected), visibility (e.g. background tasks), 2nd order effects...
3. Can we standardize metrics meters UI (e.g. like Opera)?
4. Are there cross-platform opportunities or other lessons from Lightbeam/Collusion?
5. Can we standardize more and better means to make clear privacy consequences and risk/benefit tradeoffs to end users?
6. How much of a user understanding privacy can be taken as a basis/assumption for controls

12.3.3 Next steps

Questions

1. How much commitment is there to making a change?
2. We know we have success when one can safely send a naked picture of oneself to a friend.
3. Determine use cases for privacy in context
4. Follow up with additional workshop
5. Survey business models with respect to privacy applicability (example is DNT experience)
6. Revisit P3P to reuse vocabulary definitions for JSON-LD type approach
7. Create requirements for metrics and then terms, metrics and vocabulary
8. Summarize user-centric design use cases
9. Document Best Practices for interoperability

The Workshop conclude that the questions can be addressed by the following workflow:

- identify use cases; then
- Identify streams that could contribute to metrics & identify source of information; then
- Review use cases where using these metrics for controls makes sense; then
- Implement controls

Actions

The Workshop had a session on concrete next steps. Those are less relevant to the roadmap. They were mostly a recommendation for further action within W3C. For STREWS the questions noted above are directly usable as areas for further research. The next steps are repeated here for completeness. Full details can be found in Deliverable D2.3

1. Create a best-practices document outlining type of controls that users can effectively use, reflecting experience and noting which do not work (W3C Privacy Interest Group, PING)
2. Document use cases for privacy controls in context (W3C Privacy Interest Group, PING)
3. Schedule follow up workshops to maintain community and to learn of new developments and ideas (W3C Team)
4. Comment on new architectural proposals for new Internet (e.g. pEp) to W3C submission from pEp and others (W3C TAG)
5. Survey business models with relationship to privacy concerns, alignment, issues etc (W3C Privacy Interest Group, PING)
6. Revisit the use of W3C PING vocabulary as JSON-LD vocabulary (venue undecided)
7. Identify use cases where metrics for controls make sense, identify data/information that could contribute to the metrics and identify the source of the information (W3C Privacy Interest Group, PING)
8. Create Note defining terms and vocabulary relevant to privacy controls (W3C Privacy Interest Group, PING)
9. Create a task force in the Security Interest Group to explore use cases for browser security interfaces.

12.4 Web security architectures (WWW 2015)

12.4.1 Motivation

STREWS organised a session around Web Security Architecture at the WWW Conference 2015 in Florence. We used the W3C-Track as a platform to collect feedback about Web Security Architecture from invited panellists as well as from the audience of the Conference. This allowed STREWS to validate Deliverable D1.3 in this context.

Results

The validation of D1.3 went rather smoothly. The idea about javascript compartmentalisation was floated, there was broad agreement that this is a good idea and it should be further pursued. There was no criticism, so the discussion quickly moved on to complementary measures that would help security on the Web. Which in turn yielded extraordinary results in terms of cross fertilisation from academia to standardisation.

The opening topic was the fact that trust labels might be harmful to security, as reported in Section 8.4. The paper [201] discussed was from Partner KU Leuven and presented by Nick Nikiforakis[3]. Trust labels and seals seem to be a solution to so many things. They are often a good way out in the political process. Unfortunately, those seals are harmful to security. So one of the results of the roadmap is to **not** invest further in research on labelling in the security area. Reasons included that testing is limited and thus the label reveals the vulnerabilities that

[3]Nick Nikiforakis went from KU Leuven to Stony Brook University in the meantime

were not checked. The fact of using a certain graphics makes those vulnerable services easily discoverable and attackable. And even a deeper checking revealed that the labelled services were less secure than those without a label.[4]

The second important feedback to STREWS was the criticism of bearer tokens. They are insecure and should be replaced. As the most prominent bearer tokens, Passwords, Cookies and SSO[5] - tokens were identified. The suggestion was to invest more into next generation credentials. Those should be usable, secure and robust against compromise. In the discussion, several ideas were floated. The credentials could be bound to the transport they are using. This channel binding would make key impersonation attacks harder. This is in line with the SecSess proposal that was formulated in the D1.3 deliverable.

But the final discussion was again on usability in security. And again, people were unanimous on the currently desperate state of affairs in security usability. The ways to more usability are manifold and should be subject to further research and testing.

12.5 W3C Workshops outside STREWS sponsorship

W3C had two more Workshops that are relevant for the STREWS topic, but that were covered by other projects. The W3C held a meeting on «*Next steps on trust and permissions for Web applications*» on 3-4 September 2014 sponsored by the HTML5Apps project. The meeting was held to explore further work for the Systems Applications Working Group[7]. They concluded that trusted UI, that is embedded within web apps, is an promising area for further study as it allows user actions to implicitly grant permissions in a natural way without the need for the browser to prompt the user directly. The design of the corresponding UI requires a good understanding of the use cases.

The second Workshop was the W3C Workshop on Authentication, Hardware Tokens and Beyond[22] held 10-11 September in Mountain View, California. The Workshop focused on improving secure authentication in the Open Web Platform. The Web Cryptography API defines a low-level interface to interacting with cryptographic key material that is managed or exposed by user agents. The API itself is agnostic of the underlying implementation of key storage, but provides a common set of interfaces that allow rich web applications to perform operations such as signature generation and verification, hashing and verification, encryption and decryption, without requiring access to the raw keying material. But the Web Crypto API does not give access to trusted execution environments and other hardware based security systems. The possible integration of hardware tokens was reviewed as well as other forms of authentication, with the aim of being able to replace the password with more phishing-resistant cryptographic credentials. Today on the Web, users typically use weak passwords across multiple sites, making them increasingly vulnerable to phishing and data breaches. To support high-value transactions on the Web, security and usability of user authentication must be improved. Open standards for both two-factor authentication and the use of cryptographic material for authentication would make the Web more secure. Such secure cryptographic material is naturally stored on hardware tokens such as smartcards and USB dongles that are currently difficult to interface with Web application development.

There are various initiatives already under way. FIDO[129] creates strong authentication without bundling this authentication to an identity system. But also this system is not capable of working with systems like the ones used to implement the European eIDAS Regulation[166]. The Workshop encouraged W3C to further work on the integration of such use cases into the

[4] At the same time, this topic of third-party security seals was presented and well received at the OWASP AppSec EU 2015 conference in Amsterdam. The OWASP AppSec EU conference is one of the biggest AppSec Security events in Europe, and brought together close to 600 participants from research and industry. With 5 parallel tracks (Hack, Dev, Ops, CISO and HackPra), 6 keynotes and 40 presentations, the program bundles the latest trends and techniques to (web) security practitioners in Europe.

[5] Single Sign On

Web. A first charter[210] is under consideration. The subsequent discussion on the W3C public-web-security mailinglist[6] made clear that there are still many issues ahead and that further research is needed in the area to come up with a secure connection of those worlds without creating an even higher risk for identity theft.

[6]https://lists.w3.org/Archives/Public/public-web-security/2015Sep/thread.html

Chapter 13

Standardisation Activities (DS.7)

13.1 Introduction to Standardisation

The interface between research and standardisation is a long lived discussion within the European Union. STREWS partner W3C was already part of the COPRAS [83] project that analysed how research projects in the 6th Framework programm (FP6) of the European Commission interface with Standardisation. Already in 2004, the European Commission considered standardisation a very important part of the activities of research projects. Since then, the discussion continues. COPRAS [83] revealed a so called *standardisation gap*.

Mostly, research projects will consider standardisation once they have completed their deliverables and think about further promotion and dissemination. But this takes longer than expected. The end of the project comes before the standardisation effort is completed and before any impact could have been achieved. The standardisation effort then dies down.

The COPRAS project had a lot of influence on the further discussion on the standardisation reform that finally concluded with the enacting of Regulation 1025/2012/EC [165] and the creation of the European Commission's Multistakeholder Platform for Standardisation [52]. During the reform in which the European Union tried to assess the value of standardisation in the globalised world, it was clear that especially ICT standardisation has become a key tool to implement technology at a global scale. The initial functions of standardisation like competition, consumer protection, creating interoperability (in the widest sense) have been complemented by new functions. Those include a platform for joint development of a technology for the massive scale needed by the Internet, the creation of level playing fields as a basis for further competition and the creation of entire new markets, e.g. in eCommerce. If research in ICT is also about new ideas and new innovative ways to use and further the Internet, standardisation is the tool of choice to achieve the needed technology transfer in a way that creates the level playing field needed to make this new innovative way the basis of creative further competition.

For a scientific roadmap, this has three consequences. First of all, current standardisation

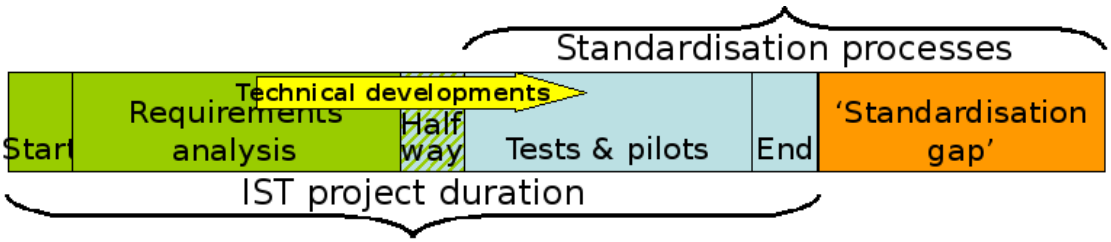


Figure 13.1: The standardisation gap in research

allows us to determine the level playing fields and features available on the Internet and on the Web in the Future. We can not say how it will exactly look like, but we have an idea about the use cases and the paradigms used. As nothing is carved in stone yet, Security research can help to avoid obvious mistakes and vulnerabilities at the very basic level of protocol and data format design. It is not without reason that every RFC in the IETF has to contain some security considerations. This requirement is more and more present also in W3C Specifications. Research, and especially security research, has to look at current technical developments and accompany those current developments. STREWS tried to lead by example. The first case study will be submitted to the various WebRTC Working Groups. A first feedback at the time of writing of this report was very positive. Secondly, some technical aspects are always true for the Web, no matter how technical implementation is done. In this case, standardisation can produce guidelines that help people implement things in a more secure way. Those guidelines can be more normative or less normative. For the more normative security specifications, they also interact with the legal system as they define the obvious technical minimum of the things one has to do to secure a given service. Third, there are the classic security specifications that show how to implement security features in an interoperable way, the most obvious topics including access control or integrity of information.

Taking those aspects into account, the current chapter can be read with various viewpoints: First with the viewpoint on where further security research for the Web is most needed or most beneficial. Secondly with a viewpoint on where we need pressure on the marketplace to actually implement certain security techniques to escape liability and third the security features we need on the Web of tomorrow.

13.2 IETF

A brief overview of the IETF and its modus operandi was previously presented in D2.1.1 and is repeated in D2.1.2 so is not included here.

The IETF does not do roadmaps, but rather the ongoing work reflects the interests of IETF participants as expressed on the various IETF mailing lists. The work of the IETF is organised by the IESG in a more reactive fashion - once there is a reasonable topic for the IETF to address with sufficient interest and with a sufficient probability of success then usually a working group is formed to tackle that topic. The goal is for most working groups to be shortlived and to close once a piece of work has been addressed. This approach does help ensure that IETF work is relevant to IETF participants but makes it harder for the IETF to be seen to be addressing broad industry or Internet trends. For example, there is, and probably won't ever be, an IETF working group on the so-called "Internet of Things" despite repeated suggestions such an activity is needed. Instead, the IETF tackle such larger trends by breaking down the problem into things where new or modified Internet Protocols are needed. So we have a working group (core[1]) defining a restful protocol called CoAP [183] and another (dice[2]) profiling the Datagram Transport Layer Security (DTLS) [73] for use in such environments.

This approach means that there is no "official" IETF roadmap so here we can only present the authors' considered opinions on relevant trends in IETF work. The authors are however well placed to provide an informed opinion on that, even if it is not authorative.

There is of course a lot of ongoing security work in the IETF that is relevant to the web, for example the WEBSEC working group [3] have almost completed work on key pinning in HTTP [76], the HTTPBIS working group [4] are nearing working group last call on HTTP/2.0 [39], and the TLS working group [5] are making good progress on TLS1.3[170]. Table 13.1 lists the current

[1] https://tools.ietf.org/wg/core/
[2] https://tools.ietf.org/wg/dice/
[3] https://tools.ietf.org/wg/websec
[4] https://tools.ietf.org/wg/httpbis/
[5] https://tools.ietf.org/wg/tls

IETF working groups that are relevant to web security and privacy and hence as background for the STREWS roadmapping activity. There are also web pages listing the full list of all current working groups[6] and also those previously concluded[7]. Recall that concluding a working group is a sign of success and not of failure as may be the case in other organisational structures.

In addition to these working groups, there are also a few IETF mailng lists that are used for discussion of possible new work, either specifically aiming at a new working group or else as a generic forum for discussion of security. Table 13.2 lists some of these. The full list of all IETF non working-group mailing lists is also available[8] and is often indicative of directions for new work.

In any normal year, the progress on HTTP and TLS etc., would be notable and positive, but in 2013-2014 in addition RFC 7258 [80] represents the high level response of the IETF to snowdonia and commits the IETF to considering the pervasive monitoring attack in all its work. That commitment should be seen as just another step in ongoing efforts [109, 110, 179, 171, 38, 56] going back two decades to improve security and privacy on the Internet. In fact, it is highly likely (and desirable) that mitigations aiming at PM will also result in general improvements in security and privacy as a not-at-all accidental side-effect. One of the most important things to consider about the PM attack is that it has motivated IETF particiants, and the Internet and Web communities more broadly, to re-consider and re-double efforts at security and those efforts migitate many forms of attack, and not just the PM attack. So even though some work may be motivated to some extent by the PM attack does not mean that that work is only about PM - in fact, mitigations are more likely to be deployed when they are effective against a broad range of attacks and not just PM.

In the near term, (which is basically what the IETF considers), the “next steps” in progressing this work (as encouraged by STREWS) involve work on cryptography, progressing the Opportunistic Security (OS) design pattern [68] and on further analysis of the PM threat.

With respect to cryptography, new co-chairs have been appointed to the IRTF’s Crrpto Forum Research Group (CFRG), partly as a result of the STRINT workshop. CFRG is currently working, at the request of the IETF TLS working group, on identifying a set of additional Elliptic curves for use with IETF protocols. There are a number of motivations for this work, with the main ones being that implementations using the new curves can significantly outperform today’s commonly used NIST curves, that such implementations are far easier to protect against side-channel attacks (mainly timing attacks), and lastly, but least significantly for many, due to some concerns that the NIST curve generation parameters were not generated in a "Nothing Up My Sleeve" (NUMS) manner. In addition, CFRG are also assisting the IETF with selecting an algorithm with RC4-like performance for cases where no AES hardware acceleration is available.

One can fairly confidently predict that these changes will also percolate out and be used by other IETF protocols and beyond. The main good impact of that should be that more performant, more side-channel resistant cryptography should be available in the near term. The hope is that that will reduce the barriers to deployment of IETF protocols, as has been seen to be the case for CloudFare’s recent TLS announcement [9] or with the startling increases in deployment of STARTTLS with SMTP over the past year. [10] [11]

That last (STARTTLS with SMTP) also provides an example of the Opportunistic Security (OS) design pattern that may help to significantly increase deployment of security mechanisms, compared to the previous IETF approach of aiming (in the author’s opinion) too high with the result that we get either no security or mutually-authenticated security and in far too many cases that means no security.

[6] https://tools.ietf.org/wg/
[7] https://tools.ietf.org/wg/concluded
[8] https://www.ietf.org/list/nonwg.html
[9] https://blog.cloudflare.com/introducing-universal-ssl/
[10] https://www.google.com/transparencyreport/saferemail/
[11] https://www.facebook.com/notes/protect-the-graph/massive-growth-in-smtp-starttls-deployment/1491049534468526

Table 13.1: Current relevant IETF working groups.

Working Group	Description
abfab	Application Bridging for Federated Access Beyond web `https://tools.ietf.org/wg/abfab/`
ace	Authentication and Authorization for Constrained Environments `https://tools.ietf.org/wg/ace/`
appsawg	Applications Area Working Group `https://tools.ietf.org/wg/appsawg/`
dane	DNS-based Authentication of Named Entities `https://tools.ietf.org/wg/dane/`
dice	DTLS In Constrained Environments `https://tools.ietf.org/wg/dice/`
httpauth	Hypertext Transfer Protocol Authentication `https://tools.ietf.org/wg/httpauth/`
httpbis	Hypertext Transfer Protocol `https://tools.ietf.org/wg/httpbis/`
jose	Javascript Object Signing and Encryption `https://tools.ietf.org/wg/jose/`
json	JavaScript Object Notation `https://tools.ietf.org/wg/json/`
oauth	Web Authorization Protocol `https://tools.ietf.org/wg/oauth/`
opsec	Operational Security Capabilities for IP Network Infrastructure `https://tools.ietf.org/wg/opsec/`
rtcweb	Real-Time Communication in WEB-browsers `https://tools.ietf.org/wg/rtcweb/`
scim	System for Cross-domain Identity Management `https://tools.ietf.org/wg/scim/`
tcpinc	TCP Increased Security `https://tools.ietf.org/wg/tcpinc/`
trans	Public Notary Transparency `https://tools.ietf.org/wg/trans/`
websec	Web Security `https://tools.ietf.org/wg/websec/`
wpkops	Web PKI OPS `https://tools.ietf.org/wg/wpkops/`

Table 13.2: Current relevant IETF non-working group mailing lists.

List name	Description/URL
dns-privacy	Discussion of DNS Privacy (expected to become DPRIVE wg) `https://www.ietf.org/mailman/listinfo/dns-privacy`
dsfjdssdfsd	Discussion of randomness in IETF protocols `https://www.ietf.org/mailman/listinfo/dsfjdssdfsd`
endymail	Discussion of elements that the IETF could do to aid in achieving better end-to-end security deployed for Internet email `https://www.ietf.org/mailman/listinfo/endymail`
perpass	List for discussion of pervasive monitoring and for triaging proposed work `https://www.ietf.org/mailman/listinfo/perpass`
pkix	List of former PKIX WG for X.509 based PKI discussion `https://www.ietf.org/mailman/listinfo/pkix`
therightkey	Discussion of future ways to find/use the right public key `https://www.ietf.org/mailman/listinfo/therightkey`
saag	General security area discussion list `https://www.ietf.org/mailman/listinfo/saag`

This OS approach, as a desirable new protocol design pattern, is however still controversial within the IETF. Some of that controversy is undoubtedly due to it being a change - not everyone is ready to admit that past approaches were significantly to blame for lack of deployment, and they do have a case to make, even if that case in no way indicates that OS will be worse (OS will be better:-). In addition, the "all-or-nothing" approach is easier to explain and also makes life easier for protocol designers. With OS, protocol designers will have to consider how and when to make security trade-offs, for example related to when to authenticate and when to accept confidentiality between unauthenticated endpoints. There is some significant education needed if the promise of the OS design pattern is to pay off.

That said, for at least TCP security, the IETF have reached consensus that a new security protocol that increases TCP security following the OS approach is a good target and a new working group (TCPINC [12]) has been formed to develop a TLS-like protocol that can secure TCP without any application layre code changes. One of the proposals being examined in TCPINC is to use tcpcrypt [41], others are based on re-using TLS primitives within TLS.

Another new working group has been proposed and is at the time of writing in the process of being discussed. The goal of this work is to provide confidentiality for DNS, again probably following the OS design pattern to an extent. As stated this work is not yet approved, but the level of interest is highly promising, given that confidentiality for DNS was never previously seriously discussed. The name for this is DPRIVE (DNS PRIVate Exchange) and the current charter text is at `https://datatracker.ietf.org/doc/charter-ietf-dprive/`. Note however, that that URL is not intended to be permanent. Should a DPRIVE working group be formed (which is highly likely in the author's opinion) the charter and documents would be at `https://tools.ietf.org/wg/dprive`.

Finally, we are still some way from fully understanding the PM threat and the mitigations that may be effective against that. In particular, IETF participants need to learn a lot more

[12] `https://tools.ietf.org/wg/tcpinc/`

about data minimisation. We have some “posterchild” cases where the lack of data minimisation in the past makes improving privacy much harder in the future. For example, in the DNS, QNAME minimisation could in principle make for a quick win for privacy. However, the fact that QNAMEs were sent in queries for decades has resulted in a number of valid (non-PM) use cases for that emerging, which now makes it hard to stop sending that data. We need to better understand the longer term impacts of adding fields to protocols both in terms of their potential for helping the PM attacker, but also in terms of the dynamics of the deployed Internet.

Overall, those trends in IETF work can fairly obviously feed into a research roadmap, encouraging work such as:

- development and analysis of new cryptographic primitives that improve performance and are more side-channel resistent, where the provenance of algorithm parameters is considered, and where industry (or the IETF community) have expressed a desire for better cryptography
- further analysis and elucidation of the OS design pattern and its implications for applications and systems security and also consideration of the possible dynamics and evolution of systems built from applications/protocols that adhere to the OS design pattern
- further analysis of the PM threat and effective but practical incremenetally deployable mitigations for PM as well as longer term approaches (more clean-slate) that might be orders of magnitude better at handling the PM threat
- analysis of the impact of most or much traffic being ciphertext - how will this affect network management and security mechanisms such as anti-spam filters, anti-virus scanners and other policy enforcement tooling that has previously assumed that cleartext was commonly available and ciphertext could be handled as an exception

13.3 W3C

13.3.1 A changing landscape

Web technologies are a foundation of many technologies. Innovation is raging on top of that platform thus challenging its robustness and resulting in a stream of questions to research and development. Instead of simple Web pages, more and more interactive elements were added over time. the Web platform is acquiring numerous additional primitives that fundamentally change its nature. Standards work on these primitives is ongoing, and is occurring in parallel with their implementation across all major browser platforms. The Open Web Platform now serves as a front-end for the distributed services using truly distributed applications with a client component delivered just-in-time. The Web is going mobile as HTML5 + JavaScript have become the platform of choice for web application development on smartphones and desktops alike. Even the so called "native" platforms often use large parts of the Web stack to transport their data across the network, but also to display information to the user and interact.

Additional challenges stem from the fact that resources on the Web are often mash-ups from several sources that form together what the end-user will see as the web page displayed. Often, this content is acquired from more than one "security domain". User agents commonly apply same-origin [33] restrictions to network requests. These restrictions prevent a client-side Web application running from one origin from obtaining data retrieved from another origin, and also limit unsafe HTTP requests that can be automatically launched toward destinations that differ from the running application’s origin. Details on the same origin policy are given in D1.1. The Web Security Guide[64].

As the Open Web develops into an application platform and extends to the mobile web, people realised that in such scenario, one can not maintain the assumption that the device is always online. But HTML documents are loaded over HTTP and traditionally fetch all of

their sub-resources via subsequent HTTP requests. This places web content at a disadvantage versus other technology stacks. Consequently a remedy was searched. There was already an element available. Web Worker [100] describes an API for running non-interactive scripts in the background in parallel to the main page. This allows for long-running scripts that are not interrupted by scripts that respond to clicks or other user interactions, and allows long tasks to be executed without yielding to keep the page responsive. But this background process still assumes an environment where the network is reachable. To complement the web worker capability with the possibility to use the background-script in an offline scenario, Service Worker creates the necessary environmental framework. It is interesting to note that Service Worker kicks in once a certain origin and a certain path a requested by a navigation event. This adds a new dimension to the attacking surface of the client device. While HTML4 + JavaScript gave applications running in a browser a very ephemeral nature, the new environment allows to have scripts that are non-transient. An attacker thus does not even have to escape the JavaScript sandbox, but can use this platform directly to achieve the goals. We can immediately see that CSP and CORS are not addressing this fully yet, as we not only have to determine where scripts and data can be loaded from, but also where data can be sent to. In the context of web applications, content is often immediately executed by the client, without giving the user a chance to approve access to sensitive or limited client resources (such as CPU and local storage). There is no separate "download", "install" and "execute" steps for a user, except for the offline world of Service Workers.

Those two use cases are only examples to show that the current trend leads to a dynamic in which the interactions between the server-side component and its client-side user agent become more flexible, powerful and complex. W3C responds to the constant challenges of the evolving Open Web Platform with a mixture of specific security development and an increased attention to security questions within the horizontal or vertical work that makes the Web work.

13.3.2 Specific security work

The security activity in W3C started with securing the XML layer with the development of XML Signature[70], XML Encryption[71] and XML Canonicalization[42]. This layer served as the basis for further security specifications within OASIS and elsewhere to secure the entire web services technology stack that is now ubiquitous in enterprise computing. The initial plan was to have an all XML Web that would use those specifications to achieve end-to-end security. But the Open Web Platform moved away from the idea of all-XML. Instead, the community opted for an evolution of the HTML specification. HTML5 + JavaScript, in turn, have different requirements and challenges with respect to security. New specific work was started and progresses steadily.

The Web Application Security Working Group[13] is developing the Content Security Policy and CSP 1.1; Cross-Origin Resource Sharing; UI Security; Secure Mixed Content; and Lightweight Isolated / Safe Content Recommendations. The goal of this work is to enable secure mash-ups, and to create a more robust Web security environment through light-weight policy expression that meshes with HTML5's built-in security policies. The group additionally aims to address clickjacking issues.

The Web Cryptography Working Group[14] is motivated by the emergence of more complex protocols executed between Web applications. The group is chartered to develop a Recommendation-track document defining an API that enables the development of such protocols. API features will include message confidentiality and authentication services, and exposing trusted cryptographic primitives from the browser. This will promote higher security on the Web as developers will no longer have to create their own or use untrusted third-party libraries for cryptographic primitives.

[13] http://www.w3.org/2011/webappsec/
[14] http://www.w3.org/2012/webcrypto/

The Web Security Interest Group[15] serves as a forum for discussion about improving standards and implementations to advance the security of the Web.

CORS - Cross-Origin Resource Sharing

There are several use cases where it is useful to overcome the same origin restrictions. This allows to share resources between different servers. This is interesting for a variety of APIs. Assuming the same-origin policy as a default does not allow any other resource to be executed, CORS[202] introduces a way to open new resources in a controlled way. This allows a web site author to securely mix content and scripts from several sources. A special care has nevertheless to be applied in order to avoid Cross-Site Request Forgery (CSRF). Especially if cross site requests are done with user credentials or the origin header in it, the specification encourages to ask for user consent, e.g. in case of OAuth [91]. Consequently, while allowing to escape the boundaries of a single origin within the Web model, the CORS specification issues numerous warnings when implementing cross-origin requests with explicit warnings concerning cross-site scripting (XSS) and CSRF. Content Security Policy (CSP)[217], presented later, addresses some of those issues.

CSP - Content Security Policy

The Content Security Policy is a mechanism that web applications can use to mitigate a broad class of content injection vulnerabilities, such as cross-site scripting (XSS). Content Security Policy is a declarative policy that lets the authors (or server administrators) of a web application inform the client about the sources from which the application expects to load resources. To mitigate XSS attacks, for example, a web application can declare that it only expects to load script from specific, trusted sources. This declaration allows the client to detect and block malicious scripts injected into the application by an attacker. Content Security Policy (CSP)[217] is not intended as a first line of defence against content injection vulnerabilities. Instead, CSP is best used as defence-in-depth, to reduce the harm caused by content injection attacks. As a first line of defence against content injection, server operators should validate their input and encode their output.

There are already three specifications on CSP: Level 1[192], updated with 1.1 and now Level 2[217]. Some of the additions and changes in CSP Level 2 are incompatible with CSP Level 1. One can see a good evolution from Level 1 to level 2. Level 1 clearly addresses the scenario of the normal average web site with a typical mash-up scenario. One can control where the scripts, the objects, the images come from. A first trial to sandbox iframes remains optional. In Level 2 the offline scenario is already present and web workers are addressed. The exchange between server and client of the policy tokens is now secured with hashes. The server computes the hash of a particular script block's contents, and includes the base64 encoding of that value in the Content-Security-Policy header. Each inline script block's contents are hashed, and compared against the whitelisted value. If there's a match, the script is executed. But this is not used for object-src, media-src etc.

CSP also contains a mechanism to report policy violations. Especially for Europe it may be interesting to come up with a specification that will correspond to the data breach notifications as required by the planned Regulation on data protection.

A new line of work is to apply pinning to the policy documents to reduce the attacking surface. CSP can only protect pages for which it is explicitly defined. Authors need to ensure that they're delivering a policy for every page on their origin in order to have confidence that a particular set of restrictions will be consistently applied. A weakness are generic error-handling pages that are constructed differently than « *real* » application pages. Those pages are sometimes forgotten in the security audit for an origin, and can offer attackers an injection vector.

CSP Pinning[214] attempts to address this concern by allowing authors to « *pin* » a baseline policy to an application's host. Conceptually, this is quite similar to the approach taken by

[15] http://www.w3.org/Security/wiki/IG

Strict Transport Security[102] and Public Key Pinning[77]. A new header, Content-Security-Policy-Pin, is defined. It instructs a user agent to remember a baseline policy that will be enforced for any document and worker delivered by an application that doesn't come with its own Content-Security-Policy header.

User Interface Security Directives for Content Security Policy

Clickjacking and phishing, commonly addressed as *UI redressing*. The User Interface Security Directives for Content Security Policy [130] tries to address this. It uses the same policy conveyance mechanism as CSP and uses headers to transport directives. It supersedes the X-Frame-Options feature. The user interface directives has input protections and a non-normative way to calculate those. Additionally, there is a feature defined for security policy violation reporting.

Subresource Integrity

New work in 2014 extends the securing of mash-ups and the integrity of the data exchanged. The very big web sites have often a network of content providers that cater to a certain web page. Authors pull scripts, images, fonts, etc. from a wide variety of services and content delivery networks, and must trust that the delivered representation is, in fact, what they expected to load. If an attacker can trick a user into downloading content from a hostile server (via DNS poisoning, or other such means), the author has no recourse. Likewise, an attacker who can replace the file on the server of a delivery network has the ability to inject arbitrary content. Delivering resources over a secure channel mitigates some of this risk: with TLS, HSTS, and pinned public keys, a user agent can be fairly certain that it is indeed speaking with the server it believes it's talking to. These mechanisms, however, authenticate only the server, not the content.

The Subresource Integrity specification [44] allows an author to pin the content to ensure the integrity of a script that loads and executes. Let's imagine one of the very popular scripts like node.js that everybody is using. node.js is loaded from numerous sources on the web. Loading it from a single source would also mean to limit the scalability and resilience of the web. But manipulating node.js has a high potential to create a large scale vulnerability. So it is important to load the right version of node.js. One way to do that is to generate a sufficiently strong cryptographic digest for a resource, and transmitting that digest to a user agent so that it may be used when fetching the resource. This, of course, requires that the necessary metadata for verification is not compromised itself. This is why the specification also requires transport security.

Future work could combine resource integrity with work that tries to leverage the DNS system, that has an existing social network and business model with securing resources attached to those DNS entries. DANE [103] does that already for TLS certificates, but one could imagine that research in the direction of registering certain content digest information into the DNS entry may help further secure mash-ups.

Referrer Policy

Under normal circumstances, a user agent (browser) sends a referrer - header with the HTTP get request. This allows the target resource to determine where the user comes from. Many user agents already allow a user to configure their user agent to suppress the sending of that referrer header. The Referrer Policy specification [74] allows the author of a web page to set a policy such that when leaving the page, the user agent does not send the referrer header to the next origin that is requested by the navigation of the user. This may sound paternalistic and has potential of abuse against tools that provide analytics and additional services around the big social networking platforms. But it also allows the social networking platforms to protect those users who are careless with the configuration of their user agent.

Mixed Content

There is another challenge when trying to secure mash-ups. While loading a page over TLS [65] the author of the page may include content from other sources that are not secured with TLS. A malicious author also may include requests to local content in a page or script. The Mixed Content specification [213] takes up mitigations that are already implemented in various user agents and lists them into one document. This concerns blocking certain resources from being fetched but also UI requirements and user controls. The work is at the beginning, thus further discussion and security research is expected in this area to explore new and unseen mitigations for browsers that are confronted with content of mixed security levels.

Confinement with Origin Web Labels (COWL)

An attacker with the ability to execute JavaScript in a web application's origin can take full control of that web application. In the typical web application scenario, untrusted JavaScript can be executed in two ways: by including it legitimately from a third party, or by having it injected through a Cross-Site Scripting vulnerability in the web application or an installed browser plugin or extension.

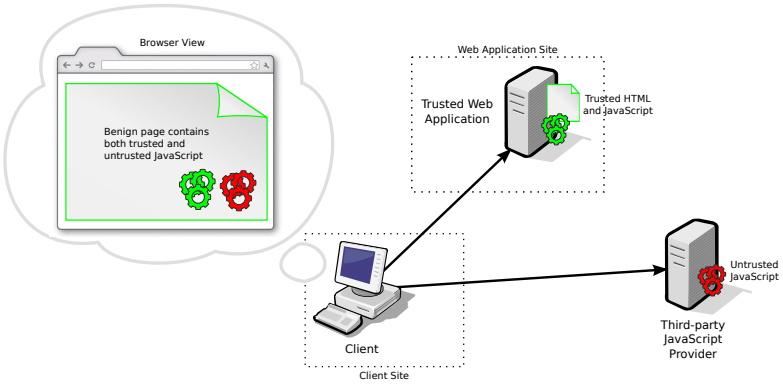


Figure 13.2: A typical web application with third-party JavaScript inclusion. The web application running in the browser combines HTML and JavaScript from a trusted source, with JavaScript from an untrusted source.

When discussing Web security, it is important to keep in mind a typical web application with third-party JavaScript and the actors involved in it. The scenario 13.2 shows such a typical web application where HTML and JavaScript from a trusted source are combined with JavaScript from an untrusted source. Remember that all JavaScript, trusted or untrusted, running in a web application's origin has access to all available resources.

STREWS' second case study on Web Security Architecture (D1.3) already proposed a further compartimentalisation. Mashups need to be controlled in a more fine grained way. In a mashup, a web application may want to specify privacy and integrity policies on data. Currently this is imagined to be done by origin labels. This way browsing contexts can be confined according to the labels and policies set. This allows web applications to share data with less trusted code and impose restrictions on how the code can further share the data involved. This comes very close to the sandboxing suggestion suggested by the Web Security Architecture. Further work on compartimentalisation and JavaScript isolation is needed.

Web Cryptography API

The Web Cryptography API defines a low-level interface to interacting with cryptographic key material that is managed or exposed by user agents. The API itself is agnostic of the underlying

implementation of key storage, but provides a common set of interfaces that allow rich web applications to perform operations such as signature generation and verification, hashing and verification, encryption and decryption, without requiring access to the raw keying material.[185]

The Web Cryptography API was developed with 7 use cases in mind. Those were:

Multi-factor Authentication A web application may wish to extend or replace existing username/password based authentication schemes with authentication methods based on proving that the user has access to some secret keying material. Rather than using transport-layer authentication, such as TLS client certificates, the web application may wish to provide a rich user experience by providing authentication within the application itself.

Using the Web Cryptography API, such an application could locate suitable client keys, which may have been previously generated via the user agent or pre-provisioned out-of-band by the web application. It could then perform cryptographic operations such as decrypting an authentication challenge followed by signing an authentication response.

The authentication data could be further enhanced by binding the authentication to the TLS session that the client is authenticating over, by deriving a key based on properties of the underlying transport.

If a user did not already have a key associated with their account, the web application could direct the user agent to either generate a new key or to re-use an existing key of the user's choosing.

Protected Document Exchange When exchanging documents that may contain sensitive or personal information, a web application may wish to ensure that only certain users can view the documents, even after they have been securely received, such as over TLS. One way that a web application can do so is by encrypting the documents with a secret key, and then wrapping that key with the public keys associated with authorized users.

When a user agent navigates to such a web application, the application may send the encrypted form of the document. The user agent is then instructed to unwrap the encryption key, using the user's private key, and from there, decrypt and display the document.

Cloud Storage When storing data with remote service providers, users may wish to protect the confidentiality of their documents and data prior to uploading them. The Web Cryptography API allows an application to have a user select a private or secret key, to either derive encryption keys from the selected key or to directly encrypt documents using this key, and then to upload the transformed/encrypted data to the service provider using existing APIs.

This use case is similar to the Protected Document Exchange use case because Cloud Storage can be considered as a user exchanging protected data with himself in the future.

Document Signing A web application may wish to accept electronic signatures on documents, in lieu of requiring physical signatures. An authorized signature may use a key that was pre-provisioned out-of-band by the web application, or it may be using a key that the client generated specifically for the web application.

The web application must be able to locate any appropriate keys for signatures, then direct the user to perform a signing operation over some data, as proof that they accept the document.

Data Integrity Protection When caching data locally, an application may wish to ensure that this data cannot be modified in an offline attack. In such a case, the server may sign the data that it intends the client to cache, with a private key held by the server. The web application that subsequently uses this cached data may contain a public key that enables it to validate that the cache contents have not been modified by anyone else.

Secure Messaging In addition to a number of web applications already offering chat based services, the rise of WebSockets and RTCWEB allows a great degree of flexibility in inter-user-agent messaging. While TLS/DTLS may be used to protect messages to web applications, users may wish to directly secure messages using schemes such as off-the-record (OTR) messaging.

The Web Cryptography API enables OTR, by allowing key agreement to be performed so that the two parties can negotiate shared encryption keys and message authentication code (MAC) keys, to allow encryption and decryption of messages, and to prevent tampering of messages through the MACs.

Javascript Object Signing and Encryption (JOSE) A web application wishes to make use of the structures and format of messages defined by the IETF Javascript Object Signing and Encryption (JOSE) Working Group. The web application wishes to manipulate public keys encoded in the JSON key format (JWK), messages that have been integrity protected using digital signatures or MACs (JWS), or that have been encrypted (JWE).

The basic API is now calling for final comments and all browsers have already started to implement. The Group is now considering works on key discovery. A recent Workshop on Authentication, Hardware Tokens and Beyond [22] has explored next steps for Web Cryptography. The Workshop was held before rechartering the Web Cryptography Working Group. It explored what the web ecosystem needs to fully realise the potential of authentication on the Web. Currently a discussion is happening in the Security Interest Group to prioritise the large amount of topics that came out of the Workshop, including authenticating the user and/or device, WebRTC authentication, credentials, trusted execution environments and more. Further features like persistent key stores have been considered. It is clear that we can expect more work in this area that is of particular interest to the banking industry and the smartcard industry.

Consequently, W3C has started to explore a consensus around starting work on hardware security. A DRAFT charter for a Hardware Security Working Group[210] was created to explore support for the creation of an API that is able to connect e.g. to a payment credential. This would allow an anonymous buyer to visit a website to shop and pay with payment credentials such as credit cards. This avoids sending credit card numbers over the wire.

13.3.3 Security considerations for horizontal work

Security challenges concern the Open Web Platform (OWP) on the one hand and the Web of Data on the other hand. The OWP is addressing certain features where security is of the utmost importance. The Web Payment Workshop [207] explored improvements of the current way to pay on the Web. The participants found a rather large set of issues, such as payment tokens, wallet interoperability, NFC, trusted interfaces and mobile payments. A further Workshop on Permissions

13.4 Other SDOs and relevant organisations

13.4.1 ECMA

Ecma International is an industry association founded in 1961 and dedicated to the standardisation of Information and Communication Technology (ICT) and Consumer Electronics (CE).

ECMAScript is the scripting language, created by TC 39[146] that is used to create web pages with dynamic behavior. ECMAScript, which is more commonly known by the name JavaScript, is an essential component of every web browser and the ECMAScript standard is one of the core standards that enable the existence of interoperable web applications on the World Wide Web.

The most current version of the 5th edition of ECMAScript from 2009. Currently the 6th edition (Harmony) is under development. It adds new security features. One of the new features is a restriction. In fact the 6th edition defines a strict variant of ECMAScript. Developers may

use this variant to restrict the usage of some features available in the language. In the interests of security error-prone features can be cut and enhanced error checking can be enabled. The strict variant also specifies additional error conditions that must be reported by throwing error exceptions in situations that are not specified as errors by the non-strict form of the language.

13.4.2 IEEE

The IEEE 802 group within the IEEE standards association sent a liaison to the STRINT workshop who subsequently worked with IESG and IAB members to sucessfully propose a new study group on privacy within IEEE 802. One initial work item for that group will be to investigate the potential for and limits that might apply to MAC address randomisation. While some vendors have already demonstrated that MAC addresses can be partly randomsed, there are issues to consider related when one consuders the full range of devices (e.g. bridges) that use MAC addresses. However, this new IEEE activity can fairly be considered an outcome from the STRINT workshop and is further described in deliverable D2.1.2.

In general, IEEE is responsible for WiFi networks and thus is working on how to secure those. Ethernet, the networking layer underlying most of the IP networks is also specified within IEEE. But IEEE also creates standards around the functions and features to be provided in intelligent electronic devices (IEDs) to accommodate critical infrastructure protection programs[1].

13.4.3 ETSI

Introduction

ETSI is a European Standards Organisation (ESO). Founded to replace the CEPT cooperation between national post and telecommunication administrations, ETSI is the place where GSM was made. Today, ETSI is the home of the 3GPP Consortium and has over 750 member organisations. We are now at the 5th generation of mobile phone and mobile data protocols that have increased the data rates significantly. All this makes ETSI a double si[15]ded SDO. On the one hand it is an international Consortium where standards for mobile communications are drawn up. And on the other hand it is an official de-jure Standards Body that executes official standardisation for the European Union.

TC - ESI

ETSI was involved in the European Electronic Signature Initiative (ESSI) and produced a range of Specifications in the area of Electronic Signatures via its TC ESI[85]. This included a joint work with W3C on XAdES[61]. The advanced electronic signatures where not integrated into browsers or services, but spawned a lot of derivative work for Web services. The work of the Web Crypto Working Group takes up this paradigm now in 2014 with its Workshop on Authentication, Hardware Tokens and Beyond[22] is trying again where XAdES failed to integrate into the Open Web Platform. ETSI is currently under way to register XAdES as a European Norm (EN) that could then be mandated by European legislation. The relation to the Web Crypto Work remains undefined.

TC - LI

The Technical Committee on Lawful Interception was created to develop a suite of standards of handover interfaces, and of rules for the carriage of technology specific interception or retained data delivery across these interfaces[15]. TC-LI operates on the basis of the EU resolution of 17th January 1995 that notes «*the legally authorized interception of telecommunications is an important tool for the protection of national interest, in particular national security and the investigation of serious crime*»[57]. This is more or less the exact opposite of what the STRINT Workshop[82] tries to achieve. The interfaces defined touch also on IP networks as defined in the

IETF. In STRINT and also in RFC 7258[80] the central argument against pervasive monitoring is not only the danger for democratic societies, but also the danger that malicious attackers will develop similar capabilities. TC LI gives those attackers a well defined interface.

13.4.4 OASIS

OASIS is a non-profit consortium that drives the development, convergence and adoption of open standards for the global information society. It promotes industry consensus and produces worldwide standards for security, Internet of Things, cloud computing, energy, content technologies, emergency management, and other areas. OASIS was founded under the name "SGML Open" in 1993. It began as a consortium of vendors and users devoted to developing guidelines for interoperability among products that support the Standard Generalized Markup Language (SGML). The consortium changed its name to "OASIS" (Organization for the Advancement of Structured Information Standards) in 1998 to reflect an expanded scope of technical work.

OASIS is very active in the B2B area, enterprise computing, commercial exchanges, XML applications and cloud services. The easy process has let to a wealth of smaller and bigger activities within OASIS. Most important for Web Security are the Security Markup Language (SAML)[153] and the eXtensible Access Control Markup Language (XACML)[141].

The Security Assertion Markup Language (SAML) defines the syntax and processing semantics of assertions made about a subject by a system entity. In the course of making, or relying upon such assertions, SAML system entities may use other protocols to communicate either regarding an assertion itself, or the subject of an assertion. The specification defines both the structure of SAML assertions, and an associated set of protocols, in addition to the processing rules involved in managing a SAML system. SAML is widely used in Web Services and SOAP-based systems, but not in any relation to the browser. SAML has to be seen in context of XACML and access to XML based high value databases.

The eXtensible Access Control Markup Language (XACML)[141] is an XML-based language for expressing and interchanging access control policies. The language offers the functionalities of most security policy languages and has standard extension points for defining new functions, data types, policy combination logic, and so on. In addition to the language, XACML defines both an architecture for the evaluation of policies and a communication protocol for message interchange. XACML is used server side, but to a lesser extend within typical Web 2.0 scenarios.

The Digital Signature Services (DSS) specifications[60] describe XML-based request/response protocols – a signing protocol and a verifying protocol. Through these protocols a client can send documents to a server and receive back a signature on the documents; or send documents and a signature to a server, and receive back an answer on whether the signature verifies the documents. This leverages the ETSI XAdES work into the B2B and commercial document exchange work that is done in OASIS. While those use nowadays IP networks, they are not part of the core Open Web Platform and thus out of scope for STREWS.

Chapter 14

STREWS Case Studies (DS.8/9)

14.1 Case Study 1: WebRTC

The STREWS WebRTC case study is reported on in deliverable D1.2 so text from that is not repeated here. In this section we provide an initial analysis of some lessons learned - more will follow as we see if or how the case study has impact on WebRTC standardisation work over the coming months and longer.

14.1.1 Set-up, execution and impact of the case study

The basic concept of attempting to get researchers and those actively involved in standards work to collaborate on a specific, timely topic was validated - we can say that following this approach is clearly possible and can produce some results. D1.2 does provide a new analysis of some aspects of WebRTC that would not otherwise have been considered from a somewhat different perspective compared to only considering the point of view of those actively developing the standards. For example, major changes to the WebRTC permissions model is not something that would be re-considered within the IETF or W3C working groups at this time, but is something that a case-study such as ours can usefully consider.

The impact of our case study is not yet known. We cannot yet say that STREWS-like case studies, nor our WebRTC study, have had significant impact. While we still believe this is quite possible, time will tell. One part of ensuring that impact that wasn't considered in the STREWS proposal was the need to translate the project output into a form useful for those developing the standards work. We have (in retrospect and unsurprisingly;-) found that sending a monolithic FP7-style deliverable to people busy developing WebRTC code and specifications is not a useful way to get them to engage on a topic. We do plan to break out some specific issues identified in D1.2 and to raise those on the IETF and W3C lists when discussion of those topics seems timely and will further report on how this works out in subsequent project reporting. (Whether at reviews or in deliverables as appropriate.)

Another lesson learned already is that the WebRTC topic was far too large for the amount of effort STREWS had available for the case study. There is simply too much work going on in the WebRTC space to fully encompass that in our case study. That is not to say that the case study was not useful, but we could have better matched the scale of effort available to the topic area, perhaps by focusing the work onto just one or a few aspects of WebRTC rather than attempting to cover the full area. However, such focus is hard to organise in an FP7 like context - one is required to propose something that proposal evaluators can evaluate a year or two before the actual work is done, which can be hard to do without being overly vague. In any case, in future, and for roadmapping purposes we could propose that case studies ought specifically consider the level of effort to be spent against the breadth of the work to be studied and ought also allow for fine-tuning of the specific case-study topic to be done late, based on developments in the

standards community and also based on the specific expertise and resources available to those carrying out the case study.

To accompany the STREWS case study, the project partners also acted as guest editors of a special issue IEEE Internet Computing Journal (Vol.18, Issue No.06), on the topic "Real-Time Communications Security on the Web", which solicited input from the research and practitioners community on the topic of WebRTC security. In this issue, the main focus of attention was on the complex problem of identity provisioning in WebRTC based applications, as this is one of the major challenges in the broad adoption this high-potential technology.

14.1.2 Identified technical challenges

Besides the identity provisioning problem (see above), the case study and the special issue resulted in the identification of the following emerging technical challenges related to WebRTC:

However, as mentioned before, this is only a subset of emerging security-related topics that require further investigation: An important aspect in securing the client-side WebRTC environment, is the ability to enforce security policies in the JavaScript execution context. As for now, the JavaScript code that controls the whole WebRTC communication runs in the browser of the end-user (and executes on his behave), but is at the same time under control of the calling page, and it will most probably include (and delegate control to) third-party libraries. Put together, this opens an an interesting attack vector for various attacks (including injection attacks) while receiving and interpreting data via the signaling or media path, and running code from the calling site and third party script JavaScript providers.

Another aspect is the overall complexity of the technology: the inherent complexity in setting up the peer-to-peer connections between browsers, and the combination of two rather distinct worlds – each with their own security model: end-to-end networking and JavaScript/Web. In particular, we expect the origin-based security model of the Web to conflict with the connection-based security model of real-time communication. For instance, questions arise as how to setup a connection across different JavaScript origins, or how to manage multiple connections and identities from within a single origin and JavaScript execution context.

Moreover, several technical solutions on the identity provisioning side as well as on the JavaScript permission side propose to enroll the end-user is making security-sensitive decisions (e.g. whether to allow the browser to set up a call with user@idp (once or permanent), whether to share his video stream or desktop, or whether to trust site A in calling external parties, ...). This will undoubtedly expose (some of) the complexity of the technology to the end-user, but even more critical this also implies that the end-to-end security of WebRTC applications will strongly depend on the judgment call of a novice end-user, potentially "promoting" this end-user to the weakest link of WebRTC.

In conclusion, while being founded on a significant body of security considerations, WebRTC will keep security-minded professionals occupied for the foreseeable future nonetheless. Given the many facets and complexity of the underlying communication and application model, this is hardly surprising. The WebRTC case also clearly illustrates how well the web is maturing: the technological advances open up a completely new area of applications, but (even more important) security is no longer an afterthought – it is already considered early in the standardisation process.

14.2 Case Study 2: Web Security Architectures

The second STREWS case study focussed on Web security architectures, as there is a clear need in the web community to build more sophisticated encapsulation and method invocation mechanisms into the DOM. Examples include Open Social (who are interested in security enhancements for the future versions of their work), Web Components (who are defining their security model), and Caja (which is being used by Google and Yahoo!). At the same time, there

are various proposals from research labs (such as ConScript, Contigo, JSand and TreeHouse) that try to define new web security architectures and confinement mechanisms for untrusted content and script.

In particular, the second case study investigated four complementary tracks within Web security architectures research:

1. **Recent Developments:** The web platform is a living ecosystem under constant change. Thus, as part of the case study, the STREWS consortium surveyed recent developments, emerging technologies and upcoming standards, that will shape the Web of tomorrow.

2. **JavaScript sandboxing:** The integration and isolation of third-party Javascript code in web applications was studied, as well as the advantages and disadvantages of deploying recently proposed sandboxing solutions

3. **Secure Session Management:** The underlying end-to-end communication between client and server was studied, and in particular the weaknesses in the session management system that undermine the security architectures on top of it.

4. **Cross-Site Scripting (XSS):** Other STREWS activities have shown (see for instance Chapter 7 (external security statistics) and Section 8.3 (empirical DOM-XSS study)), classic XSS is dominating the list of occurring security vulnerabilities and the emerging topic of client-side XSS is far from being solved. Thus, it was essential to investigate the problem domain of Cross-site Scripting (XSS), as for many proposed Web Security Architectures, an exploitable XSS vulnerability potentially fully undermines the approach's security guarantees.

As part of the case study, the security landscape of each of the tracks was surveyed in detail. Moreover, novel solutions and ideas were being proposed, as well as an exploration how these proposals can influence the Web landscape.

In the remainder of this section, we summarize the key insight gained by the case study.

14.2.1 Recent Developments

In general, recent security related developments in the Web platform can be divided into three major categories:

Security improvements to the networking and HTTP layer:

From the area of standardization, mainly driven by the IETF, various incremental and fundamental improvements to the network-portion of web applications are emerging. In the case study, the following developments were identified to be particularly noteworthy:

- *Privacy Enhanced IP Addressing* and *Transport Layer Security* improvements on the network layer,
- *DNSSEC* and *DNS Privacy* on the DNS layer,
- and *HTTP version 2* as well as *HTTP Origin Based Authentication (HOBA)* on the HTTP layer.

A major challenge for the majority of these emerging standards, is reaching significant uptake by operators of web application infrastructures, namely sever-operators and deployers of the actual web applications. Furthermore, for a subset of the proposed technologies, existing infrastructure actively hinder their adaption. E.g. despite the importance and usefulness, DNSSEC still lacks ubiquitous deployment. This is not only due to the difficulties in implementation, but also due to the fact that there are a range of middle boxes redirecting people that would be put

out of business with DNSSEC. The governmental blockings are only one examples for certain techniques that will not work any more once DNSSEC is deployed. More importantly, most Hotel networks today depend also on the manipulation of the DNS request to redirect people to the payment portal or to some click-through page that contains disclaimers and terms of service.

Server-driven security policies:

A confirmed trend in Web security technology is revolving around server driven browser enforcement. This is done via some policy document containing rules that is pushed towards the client as a part of a web application. The client is then responsible to enforce the policy correctly. On the protocol layer, *X-Frame-Options*, *HSTS* and *Key pinning* were already described. On the Web layer, there is additional room for higher level policies.

Notably in this area are the *Content Security Policy (CSP)* as well as the *User Interface Security Directives for Content Security Policy*, each addressing major client-side vulnerability class: XSS in the case of CSP and ClickJacking in the case of the User Interface Security Directives. In a similar vain, the *X-Frame-Option* header aims to protect against security threats that occur due to illegitimate framing by third parties.

Further work along the same lines is planned around *Entry Point Regulation for Web Applications*. This specification will define means to allow web applications to designate their entry points via a combination of headers and a policy manifest. User agents will be forced to restrict external navigations and other information flows into the application based on this policy to reduce the application's attack surface against Cross-Site Scripting and Cross Site Request Forgery. But Entry Point Regulation itself has some caveats as it allows the site to control deep links, bookmarks and similar features. Deep linking itself was already subject to several court decisions and is socially highly controversial. Thus Entry Point Regulation can be rather considered a dual use tool.

The user interface to security remains by far the most difficult issue to tackle. The W3C Workshop on Privacy and User Centric Controls has shown that issues related to implementing and achieving adoption related to privacy and security may be similar to those for accessibility and internationalization. The latter are seen as a constant challenge and review- point for all specifications. The Workshop has raised the right questions, but wasn't able to come up with agreed answers despite the number of high quality contributions. It has been noted earlier already that the user interface is where browsers compete. So agreement in this area is very difficult as it faces to constant challenge not to overstep the line where space is reserved for competition between the actors.

New security-centric JavaScript APIs

A multitude of new JavaScript API that either directly address security concerns or are security sensitive are currently under active development. In this field, we can identify two major clusters:

- **Secure collaboration:** APIs for *Protected Document Exchange*, *Cloud Storage*, *Document Signing*, *Data Integrity Protection* and *Secure Messaging* have the potential to enable sophisticated multi-party application scenarios. However, as it is with all newly introduced APIs, only thorough, independent security assessment, as done with the STREWS WebRTC case study, can verify that the APIs' security promises indeed hold. We foresee necessary major effort on this in the future.
- **Authentication and encryption:** The upcoming APIs for *Credential Management*, *Multi-factor Authentication* and *Web Cryptography* will bring fundamental improvement in these area to the Web browser. However, the Workshop on Web Cryptography Next Steps concluded that further work is needed to address authentication and the integration of hardware tokens. But the current Web Cryptography Working Group charter does not allow for the integration of hardware tokens. In the subsequent chartering discussion, the

Working Group decided against developing an API to access security hardware elements. This is bad news for Europe, as access from the browser and the Open Web Platform to the security features of Identity cards will be more difficult and will require dedicated software.

14.2.2 Secure Session Management

Secure session management is one of the fundamental pillars on which the security of the web platform resides which utilizes the network layer to maintain the user's authenticated state spanning both the client and the server. Recent developments have shown a broad activity in securing the information transport and authenticity of exchanges on the Web. New tools like DANE will help to determine whether a user is talking to the right service. Once the connection is established, it is important to remember what has been done and which options are available in a certain context. The shopping cart is the most prominent example of such a stateful service. This is done by session management. Session management is a crucial component in every modern web application. It links subsequent requests and temporary stateful information together, enabling a rich and interactive user experience. Unfortunately, as we will show in this section, the de facto standard cookie-based session management mechanism 2 is imperfect, which is why session management vulnerabilities rank second in the OWASP top 10 of web application vulnerabilities

As part of the second STREWS case study, the STREWS consortium conducted a comprehensive survey of proposed security improvements for web session handling and current state-of-practice countermeasures. Based on the insight won by the survey, *SecSess*, a novel secure session management was derived, as an active progress beyond the state-of-the-art.

The road from inception to adoption of security measures for the Web is difficult, aptly illustrated by the slow and cumbersome adoption process of the HttpOnly cookie attribute [63]. For SecSess, research suggests to invest in a prototype-based strategy, using concrete implementations to show the benefits and feasibility of SecSess. This implementation-driven approach allows us to maximize the opportunities to gain traction within the community and to gain momentum for further standardisation.

We have optimized exposure by presenting SecSess at industry-driven events, such as OWASP's AppSec EU conference, and workshops backed by standardisation committees, such as STREWS' STRINT workshop. Participating in these events has resulted in a promising interaction with developers from the Firefox browser and the Apache web server, both interested in providing respectively client-side and server-side support for SecSess. STREWS will try to gain attention of the Web Application Security Working Group to evaluate interest to include SecSess as one of the items for the next charter. This has to be coordinated with the relevant IETF Working Groups, as it also has protocol elements owned by the IETF.

14.2.3 JavaScript sandboxing

The case study's comprehensive overview on existing and emerging JavaScript isolation techniques showed that there is a large body of research into JavaScript sandboxing solutions, but that there is not a clear winner in terms of the perfect solution.

Browser modifications are powerful and can sandbox JavaScript efficiently, because of their prime access to the JavaScript execution environment. Unfortunately, the software modifications are difficult to distribute and maintain in the long run unless they are adopted by mainstream browser vendors.

JavaScript sandboxing mechanisms without browser modifications leverage existing browser functionality to isolate and restrict JavaScript. This approach can be slower but requires no redistribution and maintenance of browser code. In addition, it automatically works on all modern browsers.

From the review of relevant research, we can distinguish a number of components that are always present in a JavaScript sandboxing solution. By analyzing the advantages and disadvantages of these solutions, we can describe the ideal form of these components.

- **Isolation unit**

  Untrusted JavaScript should be isolated from the regular JavaScript environment so that a policy enforcement mechanism can identify on which running JavaScript code it should act.

  The isolation units used in today's JavaScript sandboxing mechanisms are not adequate: they are either not lightweight, or not free of side-effects. For instance, AdJail uses an iframe as its isolation mechanism, but does not prevent JavaScript running inside from access the DOM. JSand uses a SES sandbox as its isolation mechanism, which can mediate access to the DOM, but at a performance penalty. TreeHouse uses WebWorkers, giving executing JavaScript access to a stripped-down version of the DOM, but still allows access to some DOM functionality. Other JavaScript sandboxing mechanisms have similar disadvantages.

  Ideally, an isolation unit should be light-weight and provide a clean, side-effect free environment in which untrusted JavaScript can be run and from which it can not escape.

- **Virtual DOM and complete mediation mechanism**

  Once untrusted JavaScript code is isolated in an isolation unit, it may require access to browser functionality available in the JavaScript environment outside of its isolation unit.

  This untrusted code has certain expectations about the environment in which it executes. For instance, it may assume to have access to XMLHttpRequest functionality through certain standardized properties and functions in the DOM.

  The isolation unit must thus provide this environment, or Virtual DOM, so that any untrusted code running inside the isolation unit will not break due to expected but unavailable basic functionality.

  The Virtual DOM must be able to provide the requested functionality (e.g. XMLHttpRequest) while at the same time mediating access to it through a mediation mechanism.

  Virtual DOM implementations of today's JavaScript sandboxing mechanisms are not optimal. AdJail and TreeHouse for instance, set up a communications channel between the isolation unit and the hosting page, marshalling requests and responses between them through the postMessage() functionality. This marshalling of data inflicts a large performance hit. JSand and Object Views (in one of its forms) have no communications channel between the isolation unit and hosting page. Instead, the mediation mechanism is implemented using the membrane pattern through the Proxy API or equivalent. This approach is fast but requires wrapping the entire DOM, which can involve a lot of extra code.

  Care must be taken to never leak any unmediated references to the real DOM into the isolation unit. Such an oversight would allow the untrusted code to break out of the isolation unit.

  Ideally, browsers would provide functionality so that the Virtual DOM can automatically be wrapped using the membrane pattern in such a way that no unwrapped references to the DOM are passed to the untrusted JavaScript code.

- **Policy language**

  The mediation mechanism should be configurable through a policy, enumerating which DOM functionality that is available inside the isolation unit and describing any restrictions imposed on the usage of that functionality. For instance, a policy might dictate that access to document.location is permitted, but only to read properties and not alter them.

Such a policy language should be fine-grained and expressive enough without becoming a security problem of its own. WebJail describes an "inverse sandbox" scenario where a policy language becomes so expressive that an attacker may abuse it to steal information from other origins.

Roadmap towards adoption

Concluding from the case study's work, the STREWS consortium identifies the following roadmap towards adoption of advanced JavaScript sandboxing mechanisms:

- **Isolation unit**

  Mozilla Firefox extensions have access to functionality that allows them to execute JavaScript code in a sandbox with a selected global object and origin. This functionality, through the `Components.utils.evalInSandbox()` method, would be ideal if it was also available in the JavaScript execution environment of a web page. Unfortunately, this functionality is only available for browser extensions. On its own, `evalInSandbox()` would also not allow access mediation to a virtual DOM. Wrapping the selected global object using the Proxy API and membrane pattern could achieve this, but it is unclear whether `evalInSandbox()` works in conjunction with a Proxy object.

  Web Workers allow JavaScript code running in a web page to spawn a background thread in which JavaScript can be executed concurrently. This mechanism allows a web page to offload computationally expensive JavaScript into a separate thread, preventing the main thread from being blocked or slowed down. JavaScript code running inside a Web Worker has access to only a subset of the functions and classes available in a web page's DOM. In particular, Web Workers have no access to the DOM itself. In addition, Web Workers communicate with the main web page through postMessage, which hinders both performance and synchronous operations. An adaptation of Web Workers (e.g. a SandboxWorker) could be ideal for JavaScript sandboxing if it were possible to replace the global object inside the Web Worker with a custom object. Again, a Proxy object would allow a JavaScript sandboxing mechanism to mediate access to the DOM, if this Proxy object could replace the global object in a SandboxWorker. Because Web Workers execute in parallel with the main thread, a SandboxWorker will need to address synchronisation issues when accessing the real DOM.

- **Support for Virtual DOM creation**

  The virtual DOM needs to reflect the methods and properties available in the real DOM as accurately as possible, because running JavaScript code has certain expectations with regard to the appearance of a "standard DOM". Using the Proxy API to wrap the real DOM, presents executing JavaScript code with an apparently real DOM while at the same time allowing full mediation.

  To prevent that references to the real DOM leak from a Proxy object, the membrane pattern should be used. The implementation of this membrane pattern needs code to mimic the structure of the DOM, the DOM classes and their properties and methods. The construction of a virtual DOM thus involves a full description of the real DOM, which is not available through any JavaScript API. Such a description has to be imported from elsewhere, as JavaScript code, JSON or even IDL listings.

  Since the browser itself has the information about the structure of its own DOM, it should not be necessary to import it from elsewhere. Sharing this information from the browser in a structured way through a JavaScript API, could ease the construction of a virtual DOM.

14.2.4 Cross-Site Scripting (XSS)

Back when the Web was invented, it was intended to serve as a distribution mechanism for static, scientific documents written in HTML. With the advent of server-side scripting languages such as PHP and the introduction of client-side scripting technologies such as JavaScript, Flash or Silverlight, the Web became a fully-fledged runtime environment for sophisticated applications. In fact, operating systems such as Chrome OS or Firefox OS merely provide an execution environment for a browser such that a user can interact with web applications. As opposed to static documents, such applications frequently generate content on-the-fly based on inputs provided by the user, the user's browser, the underlying database, or other (potentially untrusted) sources.

In some cases, this input is processed by the web application in such a way that a maliciously crafted input may divert the web application's intended control flow [125, 193, 221]. Besides classical injection problems such as SQL injection [90] or Command injection [193], a considerable portion of the work focusses on so called *Cross-Site Scripting* (XSS) attacks. In the basic version of such attacks, malicious HTML tags are embedded by an attacker into Web requests issued by a client. This type of vulnerability was encountered in the late nineties by a group of Microsoft security engineers, who also coined the term [175]. The work of this group led to an advisory published by CERT in February 2000 [48], which is, to the best of our knowledge, the first known written documentation of XSS. The description given in this advisory "might nowadays be classified as the reflected/non-persistent form of the attack" [175].

The Ongoing Evolution of Cross-site Scripting

Within the last decade, a huge amount of research has been conducted to investigate the nature of such injection attacks. A multitude of different flavors of this problem emerged and our understanding of the vulnerability has fundamentally changed. This evolution can be observed best when reviewing existing research papers. The early approaches only consider reflected or persistent XSS (e.g., [40, 118, 144]). In 2005, XSS gained special attention when the infamous Samy worm [115] hit the social network MySpace and Klein published his paper on "DOM Based Cross Site Scripting or XSS of the Third Kind" [119]. Since then, a lot of different flavors of the problem emerged and, hence, the understanding of what XSS really is has changed significantly. Recent works include many other dimensions such as DOM-based XSS [119, 125], Flash-based XSS [24], mutation-based XSS [97] or CSS-based XSS [96].

XSS Today

Until today, XSS is a serious and widespread security issue. Only recently, an XSS vulnerability caused a major security breach of the Ubuntu Forum: two million user credentials were stolen, when an administrative account was hijacked via a persistent XSS attack [219].

The Open Web Application Security Project (OWASP) regularly lists XSS as one of the top three security vulnerability problems on the Web [162]. In its yearly analysis report [181], the company Whitehat Security lists XSS vulnerabilities as the most wide-spread vulnerability: according to this study, 43% of all discovered serious vulnerabilities can be accounted to XSS.

An examination of MITRE's Common Vulnerabilities & Exposures (CVE) database [140] also shows, that XSS has constantly been the second most reported vulnerability class since 2008 and even takes the top spot for 2014. In this statistic it is especially notable that no clear trend is visible, which suggests that the overall problem of XSS is in decline: Both SQL Injection and XSS expose a peak in the timeframe between 2007 and 2009, coinciding with the field of Web applications moving into the focus of the security community. But, while the frequency of SQL Injection reports has visibly declined over the following years, no such trend can be observed for the number of reports of XSS.

STREWS Survey on XSS Research

As part of the second STREWS case study, we consolidated the existing knowledge on Cross-Site Scripting, especially since the notion of this type of vulnerability obviously changed a lot over time. Our objective was to systematically review the existing literature related to XSS and present a comprehensive overview of this research field. This also enabled us to identify open problems and potential research topics that still need to be addressed. We achieved these goals by first elaborating on the different dimensions of the problem space. More specifically, we discussed the different causes behind such attacks and examine the different options of approaching the problem. We then provided a classification scheme for past, current, and future research in this area. Our classification this classification is based on the different angles from which the problem can be addressed. By applying this classification scheme to the current research landscape consisting of more than 80 papers that we reviewed, we identified the underlying trends and concepts applied to conduct, detect, mitigate, and prevent XSS vulnerabilities and attacks. Furthermore, we identified several problems that are not well-understood so far and where additional research is needed.

Open Research Problems

Our analysis also enabled us to identify open problems and topics for future research on XSS as discussed in the following.

- **Client-side XSS** As it was demonstrated in 2013 by Lekies et al [125], client-side XSS is a widespread problem in modern Web sites. In this study, the authors found DOM-based XSS vulnerabilities in roughly 10% of the Alexa Top 5000. However, only a small number of the approaches surveyed in the case study explicitly consider this specific subclass of vulnerabilities. These papers either implement variations of dynamic taint-tracking in the browser [178, 125, 143, 142] or address specific subproblems, such as second-order code injection [124] or client-side XSS caused by plugins [24]. It remains to be evaluated, to which degree the other approaches can be adapted to the specifics of client-side XSS.

- **XSS Outside of the Browser** Ever since HTML was adopted to create UIs of general purpose applications, XSS is no longer confined to the Web browser. Such attacks were for instance demonstrated for Skype [127, 28] and widgets for the Mac OS X Dashboard [174]. Lately, Web frontend technologies even found their way into mobile applications via WebViews that are embedded in the app's UI. A survey in 2012 found that up to 75% of all interviewed mobile app developers plan to use Web UI technologies in the future [195]. Apps using this technology are susceptible to XSS as well, leading to the execution of arbitrary code [145]. Up to now, only little systematic research has been conducted on XSS outside of the browser. However, given the growing importance of this development approach and the potentially enhanced privileges of injected scripts, the topic warrants further attention and should be studied in detail.

- **CSP and Script-less Attacks** The future for CSP appears to be bright. With the exception of Internet Explorer, all major browsers already support the basic CSP directives. The next iteration of the standard, CSP 1.1, will bring several enhancements for developers, such as script nonces or a JavaScript API [206] that have the potential to address current drawbacks of CSP 1.0 [208]. A strong CSP provides very robust mitigation guarantees that only break under certain circumstances, such as vulnerable server-side JavaScript generation [113] or insecure JSONP consumption [222]. Hence, CSP has the opportunity to become a major factor in the battle against XSS exploits, in case of substantial adoption of the standard in the future. However, CSP only mitigates JavaScript injection, while markup injection still remains possible. For this reason, offensive research emerged, that focus on XSS exploits that do not rely on JavaScript execution. For instance, Heiderich et al.

[96] showed how to leverage HTML/CSS injection in combination with SVG fonts to leak sensitive data from an application. Additional information exfiltration attacks that function *without* scripting have been documented by Chen et al. [49] and Zalewski [222]. The capabilities and likelihood of such attacks are still subject to future research. Furthermore, defenses against script-less attacks are needed.

- **Self-XSS** The term *Self-XSS* describes a class of social engineering attacks in which a user is tricked to copy & paste `javascript:`-URLs into the browser's address bar, causing the browser to execute the contained script in the context of the top-level document. As JavaScript's syntax allows several obfuscation tricks (e.g., encoding the full script without alpha-numeric characters [95]), the nature of the to-be-pasted URL is easily hidden and social engineering attacks are viable.

  As a response to the growing number of reported Self-XSS attacks, Facebook introduced protective measures, but remains vague about the countermeasure's technical details [78]. Furthermore, Firefox disabled the `javascript:`-URL scheme for URLs pasted in the address bar [92]. While this closes the address bar attack vector, other methods remain open, including using the Firefox Web console [148] or tricking the user to create rogue bookmarklets. None of the other major browsers have followed Firefox's example yet.

  On a technical level, Self-XSS offers the same offensive capabilities as all JavaScript injection techniques. Hence, it has to be rated on an equivalent severity level. However, up to this point only little research has been done in respect to applicable defensive measures and thus research on containing and mitigating such injections is necessary.

Chapter 15

Cyber Security (DS.10)

15.1 Introduction

In a resolution of 18 February 2003, the Council of the European Union called for a European approach towards a culture of network and information security [58]. The resolution calls for more information and for the establishment of a Cyber-Security Task Force. On 10 March 2004, ENISA was established with the Regulation 460/2004/EC[16]. It becomes clear that the European Union sees network and information security as topics that require continued attention. The reason is set out in whereas (1):

> Communication networks and information systems have become an essential factor in economic and societal development. Computing and networking are now becoming ubiquitous utilities in the same way as electricity or water supply already are. The security of communication networks and information systems, in particular their availability, is therefore of increasing concern to society not least because of the possibility of problems in key information systems, due to system complexity, accidents, mistakes and attacks, that may have consequences for the physical infrastructures which deliver services critical to the well-being of EU citizens.

The only communication networks and information systems that have become ubiquitous utilities are the phone system (including mobile), the Internet and the Web. It is clear that addressing Web Security is within the tasks that are essential to establish better network and information security in Europe.

This is reinforced by the new Regulation 526/2013/EC[59] concerning the European Union Agency for Network and Information Security (ENISA) and repealing Regulation (EC) No 460/2004. It extends the previous scope to include services.

In March 2012, Eurobarometer conducted a special study on Cyber Security[75] that contains immensely valuable data to determine prioritisation for Web Security tasks. But it also gives a hint on how large and important Web Security really is in Europe.

On 7 February 2013, the Commission and the High Representative of the European Union for Foreign Affairs and Security Policy adopted a Joint Communication to the European Parliament, the Council, the European Economic and Social Committee and the Committee of the Regions on a «*Cyber Security Strategy of the European Union: An open, safe and secure cyberspace*»[55], the Joint NIS Communication.

On the same date, the Commission adopted a proposal for a directive of the European Parliament and of the Council concerning measures to ensure a high common level of network and information security across the Union[54], the NIS-Directive.

Furthermore, the public-private network and information security (NIS) Platform [8] – NISP – was established in June 2013 to provide insight and forecast for the continued Cyber Security Strategy.

It has to be noted that in the political documents, the Web is not cited directly. Most of the documents talk about «*Cyberspace*» or «*Internet*» where in the Joint Communication[55] «*Cyberspace*» is seen as a superordinate term compared to «*Internet*», which seems rather curious to engineers. In the jungle of those terms, the report tries to extract the essence of the intended meaning for the Web despite the fact that the actual words may not directly target the Web.

But we can establish some insight for further inquiry already by analysing the NIS documents presently available.

The Cybersecurity initiative is now complemented with the eIDAS strategy for an identification and authentication scheme for the digital single market. This important work was monitored and taken into account for the roadmap. A special section takes into account the issues that eIDAS creates for Web Security and vice versa.

15.2 The Eurobarometer study

For the Eurobarometer Cybersecurity study[75] 26.593 respondents from different social and demographic groups were interviewed face-to-face at home in their mother tongue on behalf of Directorate General Home Affairs. Already in 2012 the use of the Web can be said to be ubiquitous. More and more people are using their smartphone to accomplish tasks on the move. Around half of internet users in the EU say they buy goods or services online (53%), use social networking sites (52%), or do online banking (48%), while 20% sell goods or services.

The subjective level of security seems to be quite high in the Web 2.0 context as 29% of internet users across the EU are not confident about their ability to use the internet for things like online banking or buying things online. 69% say that they are fairly or very confident. When using the internet for online banking or shopping, the two most common concerns are about someone taking or misusing personal data (mentioned by 40% of internet users in the EU) and security of online payments (38%). Most interesting for Web Security is that more than half (53%) of internet users in the EU have not changed any of their online passwords during the past year. From a security point of view, this is a very strong argument to invest more into technologies like HOBA[79]

When using the internet for online banking or shopping, the two most common concerns are about someone taking or misusing personal data (mentioned by 40% of internet users in the EU) and security of online payments (38%). This justifies the inclusion of privacy into the attacking model for STREWS. But it also tells us that the recent creation of a payment initiative within W3C is a timely reaction on real user concerns.

The regulation on data protection will establish data breach notifications. But already today, most EU citizens say they have seen or heard something about cybercrime in the last 12 months (73%), and this is most likely to have been from television (59%). Confirming Security usability as an issue, Most EU citizens do not feel very or at all well informed about the risks of cybercrime (59%) while 38% say they are very or fairly well informed. There is a clear link between being well informed and feeling confident online. More than half of those who feel confident in their ability to do online banking or buying things online say they feel well informed about cybercrime (59%). But they remain very concerned (61%) about experiencing identity theft.

Eurobarometer reports high levels of concern:

- 89% agree that they avoid disclosing personal information online;
- 74% agree that the risk of becoming a victim of cybercrime has increased in the past year;
- 72% agree that they are concerned that their online personal information is not kept secure by websites;
- 66% agree that they are concerned that information is not kept secure by public authorities.

If they experienced or were the victim of cybercrime, most respondents say they would contact the police, especially if the crime was identity theft (85%) or if they accidentally encountered child pornography online (78%). This indicates that resilience and mitigation strategies have to take users where they appear first. It means that a good strategy will enable the police to have the right communication channels to the services concerned and perhaps access to experts for Web Security.

15.3 The Joint Communication on Cyber Security Strategy

The interesting aspect in the Joint NIS Communication[55] is that it adds actual counts to the importance of Web Security. A European Digital Single Market would grow the GDP by almost €500 billion a year, which would amount to €1000 per person. The realisation of this is inhibited by the fragmentation of the market on the one hand. The cited Eurobarometer study shows concerns of users as another important inhibitor that has to be addressed.

The relevance for the Web Security Roadmap stems from the fact that the inhibited services supposed to grow the GDP are mostly services that depend on the Web. This is true for online purchases as well as for online banking that are cited as typical services concerned.

The most important contribution of the Joint NIS Communication are the principles that the Commission and the High Representative indicate to guide further work in Cybersecurity and that can be also applied for Web Security. Those frame the debate and help resolve contradicting points while evaluating the way forward. The Communication calls those «*core EU values*». Those principles are:

- Protecting fundamental rights, freedom of expression, personal data and privacy
- Access for all
- Democratic and efficient multi-stakeholder governance
- A shared responsibility to ensure security

It is remarkable, the Joint NIS Communication distinguishes between Cybersecurity and cyber resilience. This can be of some importance for the STREWS Roadmap as resilience is beyond just securing a service on the Web or preventing browsers from being manipulated. The request for resilience require the Roadmap to not only cover research directions to mitigate attacks, but to also cover research into recovering from breaches and re-establish the services affected.

Finally, research is directly addressed in the Joint NIS Communication[55]:

> R&D should fill the technology gaps in ICT security, prepare for the next generation of security challenges, take into account the constant evolution of user needs and reap the benefits of dual use technologies.

15.4 The NIS Directive

The proposed Directive concerning measures to ensure a high common level of network and information security across the Union [54] lays down measures to ensure a high common level of network and information security within the Union. It is worth noting that the proposed Directive addresses public authorities and private actors alike. It nevertheless spares so called micro-enterprises from obligations out of the Directive.

The NIS Directive first and foremost addresses public authorities and would establish a request for a range of organisational structures and communication channels. It is certainly

interesting to provide web scale technology insights into the security incident notification system. But this is rather a question of plain web technologies instead of being specific to Web security. It may though indicate further investigation in intrusion detection patterns and how those could benefit from meaningful incident reports. It has to be noted that the EDPS issued a rather critical opinion[108] with respect to the Directive by saying that the Directive fails to take a holistic view and to take into consideration other parallel initiatives, especially the proposed new regulation on data protection. With the current approach, the «*EU would risks to perpetuate a fragmented and compartmentalised approach.*» But this also depends on the interpretation of the proposed Directive.

It has to be noted that the requirements for private actors are pretty vague when the Directive says in Article 14.1:

> Member States shall ensure that public administrations and market operators take appropriate technical and organisational measures to manage the risks posed to the security of the networks and information systems which they control and use in their operations. Having regard to the state of the art, these measures shall guarantee a level of security appropriate to the risk presented. In particular, measures shall be taken to prevent and minimise the impact of incidents affecting their network and information system on the core services they provide and thus ensure the continuity of the services underpinned by those networks and information systems.

But there is no documentation for the state of the art in Web Security. The risk is that a pure procedural approach will be taken. This is known to benefit expensive consultants without improving the effective security of a service. A research agenda of the Commission could encourage the creation of a minimum state of the art document that lists vulnerabilities and types of vulnerabilities that are known to be fixed but where there is still a large amount of services that haven't fixed such vulnerability. The best example for such a situation is the SQL injection vulnerability. In research, this vulnerability is known to be fixed. So there is a awareness question. It is certainly also a good idea to invest in further research on automatic testing of services against known vulnerabilities. The Directive lacks a provision allowing for that as testing would be considered an attack and would trigger legal sanctions in most of the EU countries (with the notable exception of Germany)

Article 16 of the Directive talks about standardisation. The Directive encourages «*the use of standards and/or specifications relevant to networks and information security*». It has to be noted that most of the Cybersecurity questions touch upon technology that is specified in venues that are not European Standardisation Organisations (ESOs —ETSI/CEN/CENELEC). To address Web Security, it is certainly not very efficient to task the ESOs to come up with rules for Web Security as their community lacks the relevant expertise. In order to implement Articles 14 and 16 efficiently, the Commission will have to take the full power of its new possibilities stemming from Articles 13, 14 and 15 of the Regulation 1025/2012 on European standardisation. [165]

15.5 The NIS Platform

15.5.1 Introduction

The public-private network and information security (NIS) Platform [8] – NISP – constituted itself in general meetings in 2013. After its inception in 2013, the NIS-Platform discussed its own initial setup in plenary meetings and came up with three Working Groups:

- WG 1 will aim to identify best practice in cybersecurity risk management activities, provide guidance to enhance levels of information security and facilitate the voluntary take-up of the practices.

- WG 2 will assist and provide the groundwork for the implementation of the information sharing and incident response measures set out in the proposed NIS Directive. To do so, WG 2 will investigate the feasibility and needs to address the ability for an organisation to share cyber threat information and to utilise a standard incident management process.
- WG 3 will address issues related to Cyber Security research and innovation. WG 3 will identify the key challenges and corresponding desired outcomes in terms of innovation-focused, applied but also basic research in Cyber Security, privacy and trust; and propose new ways to promote truly multidisciplinary research that foster collaboration among researchers, industry and policy makers

All Working Groups are defined in Task and Scope by Terms of Reference that lay out the exact field of action.

15.5.2 WG 1

Unfortunately, WG 1 has not published their Roadmap on the NIS website. The Terms of Reference require WG 1 to produce an initial set of best practice guidelines by 31 May 2014, but so far, only a short with a table with completed actions has been published. None of those touch on the Open Web Platform. So those are only in so far relevant as the Web is used as a means to achieve things. So while WG 1 is working on risk assessments strategies as defined mostly in ISO (behind paywalls), there is no activity applying those risk assessment strategies on the Open Web Platform. One might consider that specific uses of the Open Web Platform may merit a specific assessment, it would be also of merit to assess those risks that are common to all uses of the Open Web Platform.

So far, WG 1 does not address Web Security at all. The STREWS Roadmap in its final iteration will hopefully help to remedy some of it. As we have seen above, the proposed NIS Directive[54] requires a Roadmap and the establishment of best practices in Articles 14 and 16. It is therefore crucial that WG 1 addresses Web Security and Internet Security in cooperation with the relevant standardisation organisations to address the implementation of the basic requirements to be established by the proposed NIS Directive.

15.5.3 WG 2

WG 2 works towards an information sharing system on Cybersecurity between EU Member states. A first document published the list of national network and information security platforms with links to further information on the sites of those national platforms. Unfortunately, this list is buried in a PDF somewhere deep in the ENISA server and not really accessible for the public at large. A future concertation may help to establish a European information platform with contact points for national response teams that is much more accessible to all.

It is interesting to see that the community in WG 2 addresses the community they know, but not the community that could change things when it comes to vulnerabilities on the Internet stack or on the Web stack. So while the Communications of the Commission, the Council and the High Representative of the European Commission is emphasising the importance of the Internet, the information system that is about to be created seems distant to the relevant actors in the field. To continue without taking the Internet or the Web into account would contribute to deepen the concerns expressed by the EDPS[108].

15.5.4 WG 3

WG 3 is closest to the work going on in STREWS as it addresses the research Roadmap. In July 2014, WG 3 published a report on the State Of The Art of Secure ICT Landscape[21]. The WG 3 approaches the roadmapping exercise alongside tooling and challenges. The document also looks into security challenges within application domains like eGovernment. While the

document contains very precious comments on Web Security, the view on Web Security is not coherent or holistic because the document structure and thus the line of thinking is much more application oriented and thus takes the platform as given. This has certain weaknesses. It is certainly true that XACML is a very famous access control language that is widely used in XML based B2B exchanges and in large middleware solutions. But it is also true that XACML is not used in the Open Web Platform. A challenge or research goal to include XACML into the Open Web Platform is not in the document of WG 3. Perhaps because the authors knew already that the Open Web Platform community is not really keen on using XACML. XACML is also not used on the mobile platform for the installing of native applications or web application.

While the research agenda talks about the future internet it starts off with classical security engineering in software that can also be seen in syssec below. More interesting from a Web Security perspective is the request for enabling methodologies and technologies:

- Secure Service Architecture and Design is addressed for the Web by STREWS case study 2
- Security Support in Programming Environments is certainly a good idea for the Web environment, but it is difficult to imagine this for the predominant JavaScript environment
- context depend security is of Interest to the Web as the same origin is setting a context and CSPs are able to overcome it
- Quantitative Aspects of Security are a good indication in web scale system. As the NESSoS report rightly states, it will help to address challenges of comprehension and transparency for the web users and their browser dashboards.

In the summary, the WG3 report acknowledges that security is not about the protection of a perimeter anymore and that the thinking in Security has to change paradigm. Having predictive monitoring as the next point in the list suggests that this could be the new paradigm. Located next to Big Data, a new paradigm becomes visible. The vast privacy problems of such an approach is not resolved. STREWS has held the STRINT workshop because the US secret services apply the paradigm of predictive monitoring on big data. This can be clearly seen from the Snowden revelations. Does it help? The Boston Marathon attacks suggest otherwise.

It is worth noting that the WG 3 report suggests diversity of devices to break the current bi-polar world, especially for hand-held devices. For Web Security considerations this does not work as all hand-held devices have implemented the Open Web Platform. One of the paradigms of the Web is that it runs on any device. We are, thus, by definition neither in a mono-culture nor in diversity. The Web is about making the diversity interoperable. WG 3 suggest the diversity to increase robustness. This reminds us that resilience and robustness are very important requirements for the interoperability between all those devices. The urgency to make the Web resilient and robust is increased by the current trend to give access to device capabilities via JavaScript APIs. In this case, the interoperability layer reaches the points the diversity was supposed to make robust.

This means there are a lot of solutions to existing problems that are not used because they are a solution to a problem but not integrated into the platform where the problem actually appears. Researchers are typically not interested in solving that issue. The have a narrow issue, which is access control. Access control is solved. The rest is not research but an implementation issue. This approach is valid for a research Roadmap for Security as such, but wouldn't be appropriate for a Roadmap on Web Security. For Web Security a given mitigation or solution only counts if it is available on the Web platform. This wasn't checked by the WG 3 Document.

Also WG 3 addresses the issue of passwords that Stephen Farrell tries to address with HOBA in the IETF[79]. But the WG 3 authors do not take that into account. As research challenges they rather see the creation of stronger authentication systems that have a higher security footprint, are not leaking information, are very usable and privacy preserving. They hint at

novel schemes like Quantum key distribution. The access control research challenges still contain anonymous credentials as the panacea. Already the PrimeLife project[186] did experiment with anonymous credentials, but the integration into the Web. The integration was very difficult and there was no decent user interface that could support the properties of a system with anonymous credentials. Anonymous credentials still merit research, but should not remove attention to the more near term solutions.

Data protection and big data are seen as challenges that will probably touch on web technologies, but data quality and provenance are out of scope for classic Web Security and would better fit under the name of data security.

The Internet of Things and cloud computing are a special area of interest to the WG 3 report. Interoperability is seen as a challenge. This was addressed by W3C in a recent Workshop[23] that tries to apply the web's large scale interoperability to sensors and machine resources. « *The Web of Things*» is addressing the issue about Interoperability of things and their communication.

15.6 The eIDAS Regulation

On 23 July 2014 the Regulation 910/2014/EC[166] on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC[163] was published in the official journal. Thus the eIDAS Regulation replaces the Directive on Electronic Signatures that proved to be unsuccessful and outdated.

The eIDAS Regulation is based on insights of research projects like STORK. It has the goal to achieve interoperability between the trust and identity systems in EU Member states. The aim is to enable a citizen of one country to use her ID given in one country to identify herself in another country. Use cases given in the introductory considerations of Regulation 910/2014/EC include transborder healtcare scenarii. This is given as an example for the numerous eGovernment services. It is supposed to help the electronic identification, authentication and signatures involving public authorities. This touches on a multitude of eGovernment services and services that require a credential from a public authority. But eIDAS is also supposed to help the establishment of union wide services as set out in the Directive 2006/123/EC[164] on services in the internal market. eIDAS creates a notification scheme for such identification and authentication schemes called « *Trust Services* ». 910/2014/EC establishes criteria for Member states wanting to register their *trust services* with the Commission and installs a liability scheme imposed on the Member state registering the « *Trust Service* » according to Art. 9 of the eIDAS Regulation.

At the MSP in June 2015, Estonia presented their authentication and eGovernment scheme that also allows for eCitizenship, meaning the use of the Estonian identification and authentication scheme in scenarii beyond the sole Estonian eGovernment applications.

To start the notification and recognition, the European Commission issued Decision 2015/296/EC[53] to implement the cooperation between Member States on the interoperability and security of electronic identification schemes. It organises the communication between Member states concerning the exchange of information, experience and best practices, a peer review process and a right to information for interoperability.

In September 2015 the Commission issued a further regulation 2015/1501/EC[167] laying down technical and operational requirements of the interoperability framework in order to ensure the interoperability of the electronic identification schemes which Member States notify to the Commission. This came together with Regulation 2015/1502/EC setting out minimum technical specifications and procedures for assurance levels for electronic identification means pursuant to Article 8(3) of Regulation 910/2014/EC[166]. For the STREWS roadmap, especially the Annex1 of the Regulation 2015/1502/EC is relevant as it defines assurance levels that are attached to clear security requirements.

The standardisation around eIDAS is mainly happening in ETSI. Current applicable standards include:

- EN 319 403 on Trust Service Provider Conformity Assessment

- EN 319 401 on General Policy requirements for *Trust Service Providers*
- EN 319 411-1 Policy and Security requirements for *Trust Service Providers* issuing certificates
- EN 319 421 on Policy and Security requirements for *Trust Service Providers* issuing time stamps
- EN 319 412 on Certificate Profiles (with profiles for natural persons, legal persons and organisations)
- EN 319 422 on a Time-stamping protocol and time-stamping profiles

None of these really help with the real integration into the Web landscape. Taking up a scenario seen in an eIDAS presentation where a student from Belgium wants to enrol electronically at a University in Torino/Italy, it is very highly likely that the enrolment process is done on a website. The main issue is now how that webpage can access the credentials held by our Belgian student. This is unsolved so far. But browser integration of the eIDAS scheme will be crucial for the success of the entire eIDAS system.

And this browser integration is far from trivial. STREWS has learned at the Workshop at the WWW conference that the use of bearer tokens is not really secure and will lead to damages. STREWS has learned from the discussion in the public-web-security mailing-list within W3C that unscoped identity tokens are a very high risk for identity theft on the Web. Using identifying tokens natively additionally creates high privacy risks and would not be able to achieve conformity with the proposed Privacy Regulation.

15.7 Conclusion

The Cybersecurity has certainly its merits and helps to organise the relevant structures and communication channels within Europe. But so far, Web Security or Internet Security are not well served. But Web technologies are crucial for the implementation of all those measures. It is therefore of utmost importance for the success of all those Cybersecurity measures to be harmonized and coordinated with the development of the Web and the Internet at large. Research and implementation play a decisive role. In WebRTC, STREWS has made a significant contribution to the overall security of the Web platform. But it is not sufficient. If the Commission wants the eIDAS scheme to succeed, it has to invest significantly into its integration into the Web and the Internet. This means participating in the work around TLS v.1.3, but also a high attention to the proposed work on Hardware Security in W3C[210]. If the Web and the Internet are not treated as a first class citizen in the eIDAS and Cybersecurity initiatives, we will see a parallel world where the measures play a certain role in enterprise computing, but are not visible or addressable by the public at large who uses the Internet and the Web. The enrolment at the University Torino will either require an expensive special device or it will be still done in paper. With investment into the Web integration, the eCommerce in the Digital single market, in turn, will receive a boost in confidence that makes European wide transactions simple and secure and will contribute greatly to growth. This includes work on connecting Web applications and trust services on a technical level and to integrate the identity scheme created by eIDAS into the Web and other Internet services, e.g. using jabber.

Bibliography

[1] 1686-2013 - ieee standard for intelligent electronic devices cyber security capabilities. Technical report.

[2] Bing Search API. `http://datamarket.azure.com/dataset/bing/search`.

[3] Common Vulnerability Scoring System (CVSS). `http://www.first.org/cvss`.

[4] Common Weakness Scoring System (CWSS). `https://cwe.mitre.org/cwss/`.

[5] Mcafee trustedsource web database. `https://www.trustedsource.org/en/feedback/url`.

[6] Network of excellence on engineering secure future internet software services and systems.

[7] *Next steps on trust and permissions for Web applications.*

[8] Nis - the network information security platform.

[9] OWASP Top Ten Project. `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`.

[10] Phantomjs: Headless webkit with javascript api. `https://www.phantomjs.org/`.

[11] Secure and trustworthy composite services.

[12] SSL Pulse. `https://www.trustworthyinternet.org/ssl-pulse/`.

[13] sslyze. `https://github.com/iSECPartners/sslyze`.

[14] Swept - securing websites through malware detection and attack prevention technologies.

[15] Terms of reference for technical committee (tc) lawful interception (li). Technical report.

[16] Regulation (ec) no 460/2004 of the european parliament and of the council of 10 march 2004 establishing the european network and information security agency. *Official Journal of the European Union*, L 077, Mar. 2004.

[17] Application vulnerability trends report: 2013. *Cenzic - the most trusted.*, 2013.

[18] 2014 data breach investigations report. 2014.

[19] Application vulnerability trends report: 2014. *Cenzic - the most trusted.*, 2014.

[20] Internet security threat report 2014. *2013 Trends, Volume 19*, 2014.

[21] State of the art of secure ict landscape. July 2014.

[22] *W3C Workshop on Authentication, Hardware Tokens and Beyond*, Sept. 2014.

[23] *W3C Workshop on the Web of Things*, June 2014.

[24] S. V. Acker, N. Nikiforakis, L. Desmet, W. Joosen, and F. Piessens. FlashOver: Automated discovery of cross-site scripting vulnerabilities in rich internet applications. In *ASIA Computer and Communications Security (ASIACCS)*, May 2012. partner: KUL; project: WebSand, NESSoS.

[25] A. Alarifi, M. Alsaleh, and A. Al-Salman. Security analysis of top visited arabic web sites. In *Advanced Communication Technology (ICACT), 2013 15th International Conference on*, pages 173–178. IEEE, 2013.

[26] Apache Software Foundation. Apache tomcat - migration guide. *Online at* `http://tomcat.apache.org/migration-7.html`, 2013.

[27] L. Arsene. Xbox Live Accounts of Microsoft Employees Hacked Using Social Engineering. `http://www.hotforsecurity.com/blog/xbox-live-accounts-of-microsoft-employees-hacked-using-social-engineering-5716.html`.

[28] A. Asher. Explaining the cross site scripting bug in skype for windows. [online], `http://blogs.skype.com/2011/07/15/explaining-the-cross-site-scri/`, July 2011.

[29] M. Balduzzi, C. T. Gimenez, D. Balzarotti, and E. Kirda. Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications. In *18th Annual Network and Distributed System Security Symposium*, San Diego, USA, 2011.

[30] R. Barnes, B. Schneier, C. Jennings, and T. Hardie. Pervasive Attack: A Threat Model and Problem Statement. Internet-Draft draft-barnes-pervasive-problem-01, Internet Engineering Task Force, July 2014. Work in progress.

[31] R. Barnes, B. Schneier, C. Jennings, T. Hardie, B. Trammell, C. Huitema, and D. Borkmann. Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement. Internet-Draft draft-iab-privsec-confidentiality-threat-07, Internet Engineering Task Force, May 2015. Work in progress.

[32] A. Barth. HTTP state management mechanism. *IETF RFC*, 2011.

[33] A. Barth. The Web Origin Concept. RFC 6454 (Proposed Standard), Dec. 2011.

[34] A. Barth, C. Jackson, and J. C. Mitchell. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 75–88, New York, NY, USA, 2008. ACM.

[35] D. Bates, A. Barth, and C. Jackson. Regular expressions considered harmful in client-side XSS filters. In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 91–100, New York, NY, USA, 2010. ACM.

[36] K. Beckers, P. Philippaerts, and F. Martinelli. D4.3 part ii: Engineering secure future internet services: A research manifesto and agenda from the nessos community- final release. Technical report.

[37] F. Belanger, J. S. Hiller, and W. J. Smith. Trustworthiness in electronic commerce: the role of privacy, security, and site attributes. *The Journal of Strategic Information Systems*, 11(3):245–270, 2002.

[38] S. Bellovin and R. Housley. Guidelines for Cryptographic Key Management. RFC 4107 (Best Current Practice), June 2005.

[39] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). Internet-Draft draft-ietf-httpbis-http2-17, Internet Engineering Task Force, Feb. 2015. Work in progress.

[40] P. Bisht and V. N. Venkatakrishnan. Xss-guard: Precise dynamic prevention of cross-site scripting attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, DIMVA '08, pages 23–43, Berlin, Heidelberg, 2008. Springer-Verlag.

[41] A. Bittau, D. Boneh, M. Hamburg, M. Handley, D. Mazieres, and Q. Slack. Cryptographic protection of TCP Streams (tcpcrypt). Internet-Draft draft-bittau-tcpinc-01, Internet Engineering Task Force, July 2014. Work in progress.

[42] J. Boyer, G. Marcy, et al. Canonical XML Version 1.1. Technical report.

[43] F. Braun, D. Akhawe, J. Weinberger, and M. West. Subresource Integrity. W3C Working Draft, W3C, Mar. 2014. Work in progress. `http://www.w3.org/TR/2014/WD-SRI-20140318/`.

[44] F. Braun, D. Akhawe, J. Weinberger, and M. West. Subresource Integrity. W3C Working Draft, W3C, Sept. 2015. Work in progress. `http://www.w3.org/TR/2015/WD-SRI-20150916/`.

[45] P. Bright. Anonymous speaks: the inside story of the HBGary hack. `http://arstechnica.com/tech-policy/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack/`.

[46] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic Clustering of the Web. *Computer Networks and ISDN Systems*, 29(8-13), Sept. 1997.

[47] D. Canali, D. Balzarotti, and A. Francillon. The role of web hosting providers in detecting compromised websites. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 177–188, 2013.

[48] CERT. Advisory ca-2000-02 malicious html tags embedded in client web requests, February 2000.

[49] E. Y. Chen, S. Gorbaty, A. Singhal, and C. Jackson. Self-Exfiltration: The Dangers of Browser-Enforced Information Flow Control. In *IEEE Symposium on Security and Privacy*, 2012.

[50] P. Chen, N. Nikiforakis, L. Desmet, and C. Huygens. A dangerous mix: Large-scale analysis of mixed-content websites. In *Proceedings of the 16th Information Security Conference (ISC 2013)*, 2013.

[51] P. Chen, N. Nikiforakis, C. Huygens, and L. Desmet. A Dangerous Mix: Large-scale analysis of mixed-content websites. In *Proceedings of the 16th Information Security Conference*, ISC '13, Dallas, USA, 2013.

[52] E. Commission. European multi stakeholder platform on ict standardisation, Nov. 2011.

[53] E. Commission. *Commission Implementing Decision (EU) 2015/296 of 24 February 2015 establishing procedural arrangements for cooperation between Member States on electronic identification pursuant to Article 12(7) of Regulation (EU) No 910/2014 of the European Parliament and of the Council on electronic identification and trust services for electronic transactions in the internal market*, volume 53, pages 14–20. feb 2015.

[54] T. E. Commission. Proposal for a directive of the european parliament and of the council concerning measures to ensure a high common level of network and information security across the union. *COM*, 2013(48).

[55] T. E. Commission and T. H. R. of the Union. Joint communication to the european parliament, the council, the european economic and social committee and the committee of the regions cybersecurity strategy of the european union: An open, safe and secure cyberspace. *JOIN*, 2013(01), Feb. 2013.

[56] A. Cooper, H. Tschofenig, B. Aboba, J. Peterson, J. Morris, M. Hansen, and R. Smith. Privacy Considerations for Internet Protocols. RFC 6973 (Informational), July 2013.

[57] T. E. Council. Council resolution of 17 january 1995 on the lawful interception of telecommunications. *Official Journal C*, 39(329), 04 November 1996.

[58] T. E. Council. Council resolution of 18 february 2003 on a european approach towards a culture of network and information security. *Official Journal of the European Union*, C 48:01–02, Feb. 2003.

[59] T. E. Council. Regulation (eu) no 526/2013 of the european parliament and of the council of 21 may 2013 concerning the european union agency for network and information security (enisa) and repealing regulation (ec) no 460/2004. *Official Journal L*, 165:41–58, June 2013.

[60] J. C. Cruellas-Ibarz. Digital signature service core protocols, elements, and bindings. Technical report.

[61] J. C. Cruellas-Ibarz. Electronic signatures and infrastructures; profiles of xml advanced electronic signatures based on ts 101 903 (xades). Technical report.

[62] P. De Ryck, L. Desmet, P. Philippaerts, and F. Piessens. A security analysis of next generation web standards. Technical report, European Network and Information Security Agency (ENISA), July 2011.

[63] P. De Ryck, L. Desmet, F. Piessens, and M. Johns. *Primer on Client-Side Web Security*. SpringerBriefs in Computer Science. Springer, 2014.

[64] P. De Ryck, W. Joosen, F. Piessens, M. Johns, E. Davies, B. Bos, T. Roessler, L. Desmet, S. Lekies, J. T. Mühlberg, and S. Farrell. Web-platform security guide: Security assessment of the web ecosystem. Technical report.

[65] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627.

[66] A. Doupé, M. Cova, and G. Vigna. Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners. In *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, Bonn, Germany, July 2010.

[67] C. Drake. The superfecta report special edition - 2013 year in review - cyberattack trend analysis. 2014.

[68] V. Dukhovni. Opportunistic Security: Some Protection Most of the Time. Internet-Draft draft-dukhovni-opportunistic-security-06, Internet Engineering Task Force, Nov. 2014. Work in progress.

[69] T. Duong and J. Rizzo. *Here Come The ⊕ Ninjas*, 2011.

[70] D. Eastlake et al. XML Signature Syntax and Processing Version 1.1. Technical report.

[71] D. Eastlake et al. XML Signature Syntax and Processing Version 1.1. Technical report.

[72] B. Edelman. Adverse Selection in Online "Trust" Certifications. In *Proceedings of the 11th International Conference on Electronic Commerce*, ICEC '09, pages 205–212, 2009.

[73] L. Eggert. Moving the Undeployed TCP Extensions RFC 1072, RFC 1106, RFC 1110, RFC 1145, RFC 1146, RFC 1379, RFC 1644, and RFC 1693 to Historic Status. RFC 6247 (Informational), May 2011.

[74] J. Eisinger and M. West. Referrer Policy. *W3C Working Draft*, 2014.

[75] Eurobarometer. Special eurobarometer 390 cyber security. *Special Eurobarometer*, 390, July 2012.

[76] C. Evans, C. Palmer, and R. Sleevi. Public Key Pinning Extension for HTTP. Internet-Draft draft-ietf-websec-key-pinning-21, Internet Engineering Task Force, Oct. 2014. Work in progress.

[77] C. Evans, C. Palmer, and R. Sleevi. Public Key Pinning Extension for HTTP. RFC 7469 (Proposed Standard), Apr. 2015.

[78] Facebook. Keeping you safe from scams and spam. [online], `https://www.facebook.com/notes/facebook-security/keeping-you-safe-from-scams-and-spam/10150174826745766`, May 2011.

[79] S. Farrell, P. Hoffman, and M. Thomas. HTTP Origin-Bound Authentication (HOBA). *IETF Internet Draft*, 2013.

[80] S. Farrell and H. Tschofenig. Pervasive Monitoring Is an Attack. RFC 7258 (Best Current Practice), May 2014.

[81] S. Farrell, R. Wenning, B. Bos, M. Blanchet, and H. Tschofenig. Report from the Strengthening the Internet (STRINT) workshop. Internet-Draft draft-iab-strint-report-03, Internet Engineering Task Force, Sept. 2015. Work in progress.

[82] S. Farrell, R. Wenning, and H. Tschofenig. W3c/iab workshop on strengthening the internet against pervasive monitoring (strint).

[83] C. E. C. for Standardization, E. E. T. S. Institute, T. O. Group, W. T. W. W. W. Consortium, and C. E. C. E. for Standardization. Cooperation platform for research and standards (copras).

[84] S. Fox. Pew Research : 51% of U.S. Adults Bank Online. `http://www.pewinternet.org/2013/08/07/51-of-u-s-adults-bank-online/`.

[85] R. Genghini and E. Giessmann. Etsi technical committee on electronic signatures and infrastructures.

[86] Google Developers. Chrome Extensions - Developer's Guide. [online], `http://developer.chrome.com/extensions/devguide.html`, last access 06/05/13, 2012.

[87] J. Grossman. Whitehat website security statistics report. *WhiteHat Security*, 2011.

[88] J. Grossman. Whitehat website security statistics report. *WhiteHat Security*, 2012.

[89] J. Grossman. Whitehat website security statistics report. *WhiteHat Security*, 2013.

[90] W. Halfond, J. Viegas, and A. Orso. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA*, pages 13–15, 2006.

[91] E. Hammer-Lahav, D. Recordon, and D. Hardt. The oauth 2.0 authorization protocol. *Network Working Group Internet-Draft*, 2011.

[92] N. Hammond. Social Engineering Issue with "javascript:" URLs. Mozilla Bugzilla bug bug #527530, 2009.

[93] M. M. Head and K. Hassanein. Trust in e-commerce: evaluating the impact of third-party seals. *Quarterly Journal of Electronic Commerce*, 3:307–326, 2002.

[94] OpenSSL 'Heartbleed' vulnerability (CVE-2014-0160). `https://www.us-cert.gov/ncas/alerts/TA14-098A`.

[95] M. Heiderich, E. Nava, G. Heyes, and D. Lindsay. *Web Application Obfuscation: -/WAFs..Evasion..Filters//alert (/Obfuscation/)-*. Elsevier/Syngress, 2010.

[96] M. Heiderich, M. Niemietz, F. Schuster, T. Holz, and J. Schwenk. Scriptless attacks: stealing the pie without touching the sill. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 760–771, 2012.

[97] M. Heiderich, J. Schwenk, T. Frosch, J. Magazinius, and E. Z. Yang. mxss attacks: Attacking well-secured web-applications by using innerhtml mutations. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[98] D. Hepper. Gmail CSRF vulnerability explained. *Online at `http://daniel.hepper.net/blog/2008/11/gmail-csrf-vulnerability-explained/`*, 2008.

[99] G. Heyes. Bypassing XSS Auditor. [online], `http://www.thespanner.co.uk/2013/02/19/bypassing-xss-auditor/`, last accessed 08/05/13, February 2013.

[100] I. Hickson. Web Workers. *W3C Candidate Recommendation*, 2012.

[101] J. Hodges, C. Jackson, and A. Barth. HTTP strict transport security (HSTS). *IETF RFC*, 2012.

[102] J. Hodges, C. Jackson, and A. Barth. HTTP Strict Transport Security (HSTS). RFC 6797 (Proposed Standard), Nov. 2012.

[103] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698 (Proposed Standard), Aug. 2012. Updated by RFC 7218.

[104] J. Horrigan. Pew Research : Online Shopping. `http://www.pewinternet.org/2008/02/13/online-shopping/`.

[105] R. W. Houston and G. K. Taylor. Consumer Perceptions of CPA WebTrust assurances: Evidence of an Expectation Gap. *International Journal of Auditing*, 3(2):89–105, 1999.

[106] X. Hu, Z. Lin, and H. Zhang. Trust promoting seals in electronic markets: an exploratory study of their effectiveness for online sales promotion. *Journal of Promotion Management*, 9(1-2):163–180, 2002.

[107] T. Hunt. Why I am the worldâĂŹs greatest lover (and other worthless security claims). `http://www.troyhunt.com/2013/05/why-i-am-worlds-greatest-lover-and.html`.

[108] P. Hustinx. Opinion of the european data protection supervisor on the joint communication of the commission and of the high representative of the european union for foreign affairs and security policy on a 'cyber security strategy of the european union: An open, safe and secure cyberspace', and on the commission proposal for a directive concerning measures to ensure a high common level of network and information security across the union.

[109] IAB and IESG. IAB and IESG Statement on Cryptographic Technology and the Internet. RFC 1984 (Best Current Practice), Aug. 1996.

[110] IAB and IESG. IETF Policy on Wiretapping. RFC 2804 (Informational), May 2000.

[111] J. James and S. MacDougall. Using McAfee secure/trustguard as attack tools. In *Derby-Con*, 2012.

[112] J. P. John, F. Yu, Y. Xie, A. Krishnamurthy, and M. Abadi. deSEO: Combating Search-result Poisoning. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, 2011.

[113] M. Johns. PreparedJS: Secure Script-Templates for JavaScript. In *10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA '13)*, LNCS. Springer, July 2013.

[114] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic. Secubat: a web vulnerability scanner. In *Proceedings of the 15th international conference on World Wide Web*, pages 247–256. ACM, 2006.

[115] S. Kamkar. I'll never get caught. i'm popular!, April 2005.

[116] D. J. Kim, C. Steinfield, and Y.-J. Lai. Revisiting the role of web assurance seals in business-to-consumer electronic commerce. *Decision Support Systems*, 44(4):1000–1015, 2008.

[117] K. M. Kimery and M. McCord. Third-party assurances: the road to trust in online retailing. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. IEEE, 2002.

[118] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *ACM Symposium On Applied Computing (SAC)*, SAC '06, pages 330–337, New York, NY, USA, 2006. ACM.

[119] A. Klein. Dom based cross site scripting or xss of the third kind. *Web Application Security Consortium, Articles*, 4, 2005.

[120] E. Kovacs. CSRF Vulnerability in eBay Allows Hackers to Hijack User Accounts. *Online at* `http://news.softpedia.com/news/CSRF-Vulnerability-in-eBay-Allows-Hackers-to-Hijack-User-Accounts-Video-383316.shtml`, 2013.

[121] B. Krebs. A First Look at the Target Intrusion, Malware. `http://krebsonsecurity.com/2014/01/a-first-look-at-the-target-intrusion-malware/`.

[122] B. Krebs. Adobe Breach Impacted At Least 38 Million Users. `http://krebsonsecurity.com/2013/10/adobe-breach-impacted-at-least-38-million-users/`.

[123] A. Langley. Apple's SSL/TLS Bug. `https://www.imperialviolet.org/2014/02/22/applebug.html`.

[124] S. Lekies and M. Johns. Lightweight integrity protection for web storage-driven content caching. *Web 2.0 Security and Privacy (W2SP)*, 2012.

[125] S. Lekies, B. Stock, and M. Johns. 25 million flows later-large-scale detection of dom-based xss. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[126] S. Lekies, B. Stock, and M. Johns. 25 million flows later: large-scale detection of dom-based xss. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1193–1204. ACM, 2013.

[127] M. Lucinskij. Skype videomood XSS. Posting to the full disclosure mailinglist, `http://seclists.org/fulldisclosure/2008/Jan/0328.html`, January 2008.

[128] R. Lundeen, J. Ou, and T. Rhodes. New ways i'm going to hack your web app. 2011.

[129] S. Machani, R. Philpott, S. Sampath, J. Kemp, and J. Hodges. FIDO UAF Architectural Overview. Fido alliance proposed standard, dec 2014.

[130] G. Maone, D. L.-S. Huang, T. Gondrom, and B. Hill. User Interface Security Directives for Content Security Policy. W3C Working Draft, W3C, Mar. 2014. Work in progress. `http://www.w3.org/TR/2014/WD-UISecurity-20140318/`.

[131] E. Markatos and D. Balzarotti. syssec Network of Excellence.

[132] M. Marlinspike. New tricks for defeating ssl in practice. *Blackhat*, 2009.

[133] M. Marlinspike. Sslstrip. *Online at http://www.thoughtcrime.org/software/sslstrip/*, 2009.

[134] B. Martin, M. Brown, A. Paller, and S. Christey. Cwe/sans top 25 most dangerous programming errors. *Online at http://cwe.mitre.org/top25/pdf/2009_cwe_sans_top_25.pdf*, 2009.

[135] McAfee. The Economic Impact of Cybercrime and Cyber Espionage. `http://www.mcafee.com/sg/resources/reports/rp-economic-impact-cybercrime.pdf`.

[136] R. J. McCullen. 2012 global security report - we tested, we analyzed, we discovered. *Trustwave Global Security Report*, 2012.

[137] R. J. McCullen. 2013 global security report - we tested, we analyzed, we discovered. *Trustwave Global Security Report*, 2013.

[138] R. J. McCullen. 2014 global security report - we tested, we analyzed, we discovered. *Trustwave Global Security Report*, 2014.

[139] Microsoft. IE8 Security Part IV: The XSS Filter. 2008.

[140] MITRE. Common vulnerabilities and exposures - the standard for information security vulnerability names.

[141] T. Moses. eXtensible Access Control Markup Language (XACML) v2.0, 2005.

[142] R. Mui and P. Frankl. Preventing web application injections with complementary character coding. In *European Symposium on Research in Computer Security (ESORICS)*, ESORICS'11, pages 80–99, Berlin, Heidelberg, 2011. Springer-Verlag.

[143] Y. Nadji, P. Saxena, and D. Song. Document structure integrity: A robust basis for cross-site scripting defense. In *Symposium on Network and Distributed System Security (NDSS)*, 2009.

[144] F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Cross-site scripting prevention with dynamic data tainting and static analysis. In *Symposium on Network and Distributed System Security (NDSS)*, 2007.

[145] M. Neugschwandtner, M. Lindorfer, and C. Platzer. A View to a Kill: WebView Exploitation. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2013.

[146] J. Neumann et al. Tc 39 ecmascript.

[147] N. Nikiforakis. Bypassing Chrome's Anti-XSS filter. [online], `http://blog.securitee.org/?p=37`, last access 08/05/13, September 2011.

[148] N. Nikiforakis. Firefox and self-xss. [online], `http://blog.securitee.org/?p=114`, January 2012.

[149] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. You are what you include: Large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 736–747, 2012.

[150] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *IEEE Security and Privacy*, 2013.

[151] N. Nikiforakis, Y. Younan, and W. Joosen. HProxy: Client-side detection of SSL stripping attacks. In *Proceedings of the 7th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA'10)*, 2010.

[152] N. Olivarez-Giles. Snapchat Data Breach Exposes Millions of Names, Phone Numbers. `http://blogs.wsj.com/digits/2014/01/01/snapchat-alleged-leak-4-million-users/`.

[153] Organization for the Advancement of Structured Information Standards. Security Assertion Markup Language (SAML) v2.0, 2005.

[154] OWASP. Authentication cheat sheet.

[155] OWASP. Buffer overflow attack.

[156] OWASP. Content spoofing.

[157] OWASP. Cross-site request forgery (csrf).

[158] OWASP. Cross-site scripting (xss).

[159] OWASP. Information leakage.

[160] OWASP. Session management cheat sheet.

[161] OWASP. Sql injection.

[162] OWASP. Cross-site scripting (xss), September 2013.

[163] E. Parliament and E. Council. *Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures*, volume 013, pages 12–20. jan 2000.

[164] E. Parliament and E. Council. *Directive 2006/123/EC of the European Parliament and of the Council of 12 December 2006 on services in the internal market*, volume 376, pages 36–68. dec 2006.

[165] E. Parliament and E. Council. *Regulation (EU) No 1025/2012 of the European Parliament and of the Council of 25 October 2012 on European standardisation*, pages 12–33. Number 316. Nov. 2012.

[166] E. Parliament and E. Council. *Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC*, volume 257, pages 73–114. Aug. 2014.

[167] E. Parliament and E. Council. *Commission Implementing Regulation (EU) 2015/1501 of 8 September 2015 on the interoperability framework pursuant to Article 12(8) of Regulation (EU) No 910/2014 of the European Parliament and of the Council on electronic identification and trust services for electronic transactions in the internal market*, volume 235, pages 1–6. sep 2015.

[168] M. Rajab, L. Ballard, N. Jagpal, P. Mavrommatis, D. Nojiri, N. Provos, and L. Schmidt. Trends in circumventing web-malware detection. *Google, Google Technical Report*, 2011.

[169] D. Reisinger. Syrian Electronic Army hacks Forbes, steals user data. `http://www.cnet.com/news/syrian-electronic-army-hacks-forbes-steals-user-data/`.

[170] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-tls13-08, Internet Engineering Task Force, Aug. 2015. Work in progress.

[171] E. Rescorla and B. Korver. Guidelines for Writing RFC Text on Security Considerations. RFC 3552 (Best Current Practice), July 2003.

[172] M. Riguidel. Final recommendations report on future global research challenges in ict trust and security. Technical report.

[173] J. Rizzo and T. Duong. Crime: Compression ratio info-leak made easy. In *ekoparty Security Conference*, 2012.

[174] T. Roessler. When Widgets Go Bad. Lightning talk at the 24C3 conference, `http://log.does-not-exist.org/archives/2007/12/28/2160_when_widgets_go_bad.html`, December 2007.

[175] D. Ross. Happy 10th birthday cross-site scripting!, December 2009.

[176] D. Ross and T. Gondrom. HTTP Header X-Frame-Options. *IETF RFC*, 2013.

[177] P. Saxena, S. Hanna, P. Poosankam, and D. Song. FLAX: Systematic Discovery of Client-side Validation Vulnerabilities in Rich Web Applications. In *NDSS*. The Internet Society, 2010.

[178] P. Saxena, S. Hanna, P. Poosankam, and D. Song. Flax: Systematic discovery of client-side validation vulnerabilities in rich web applications. In *Symposium on Network and Distributed System Security (NDSS)*. The Internet Society, 2010.

[179] J. Schiller. Strong Security Requirements for Internet Engineering Task Force Standard Protocols. RFC 3365 (Best Current Practice), Aug. 2002.

[180] M. Schunter and F. Hirsch, editors. *W3C Workshop on Privacy and User-Centric Controls*, 2014.

[181] W. Security. Website security statistics report, May 2013.

[182] D. Sellers. ASP.NET 2.0 and the new HTTP-only property. *MSDN Blogs*, March 2006.

[183] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252 (Proposed Standard), June 2014.

[184] R. Siles. Session management cheat sheet - session id properties. *Online at https://www.owasp.org/index.php/Session_Management_Cheat_Sheet#Session_ID_Properties*, 2013.

[185] R. Sleevi and M. Watson. Web Cryptography API. W3C Candidate Recommendation, W3C, Dec. 2014. Work in progress. http://www.w3.org/TR/2014/CR-WebCryptoAPI-20141211/.

[186] D. Sommer and J. Camenisch. The primelife project.

[187] S. Son and V. Shmatikov. The postman always rings twice: Attacking and defending postmessage in html5 websites.

[188] S. Stamm, B. Sterne, and G. Markham. Reining in the web with content security policy. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 921–930, New York, NY, USA, 2010. ACM.

[189] E. Stark, M. Hamburg, and D. Boneh. Symmetric cryptography in javascript. In *ACSAC '09: Proceedings of the 2009 Annual Computer Security Applications Conference*, pages 373–381, Washington, DC, USA, 2009. IEEE Computer Society.

[190] B. Sterne and A. Barth. Content Security Policy 1.0. *W3C Candidate Recommendation*, 2012.

[191] B. Sterne and A. Barth. Content Security Policy 1.0. W3C Candidate Recommendation, W3C, Nov. 2012. http://www.w3.org/TR/2012/CR-CSP-20121115/.

[192] B. Sterne and A. Barth. Content Security Policy 1.0. W3C Candidate Recommendation, W3C, Nov. 2012. http://www.w3.org/TR/2012/CR-CSP-20121115/.

[193] Z. Su and G. Wassermann. The essence of command injection attacks in web applications. *SIGPLAN Not.*, 41(1):372–382, Jan. 2006.

[194] Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications. In *Proceedings of POPL'06*, January 2006.

[195] D. Taft. 75% of Developers Using HTML5. [online], http://www.eweek.com/c/a/Application-Development/75-of-Developers-Using-HTML5-Survey-508096/, 2012.

[196] D. Talbot. What should you do about heartbleed? excellent question. *MIT Technology Review*, Apr. 2014. Available at http://www.technologyreview.com/view/526406/what-should-you-do-about-heartbleed-excellent-question/.

[197] The Guardian. Edward Snowden. *Online at http://www.theguardian.com/world/edward-snowden*, 2013.

[198] Trend Micro Site Safety Center. http://global.sitesafety.trendmicro.com/.

[199] S. Van Acker, N. Nikiforakis, L. Desmet, F. Piessens, and W. Joosen. Monkey-in-the-browser: Malware and vulnerabilities in augmented browsing script markets. In *ASIACCS*, June 2014.

[200] T. van Goethem, P. Chen, N. Nikiforakis, L. Desmet, and W. Joosen. Large-scale security analysis of the web: Challenges and findings. In T. Holz and S. Ioannidis, editors, *Trust and Trustworthy Computing - 7th International Conference, TRUST 2014, Heraklion, Crete, Greece, June 30 - July 2, 2014. Proceedings*, volume 8564 of *Lecture Notes in Computer Science*, pages 110–126. Springer, 2014.

[201] T. van Goethem, F. Piessens, W. Joosen, and N. Nikiforakis. Clubbing seals: Exploring the ecosystem of third-party security seals. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM Conference on Computer and Communications Security*, pages 918–929. ACM, 2014.

[202] A. van Kesteren. Cross-Origin Resource Sharing. W3C Recommendation, W3C, Jan. 2014. `http://www.w3.org/TR/2014/REC-cors-20140116/`.

[203] M. Vasek and T. Moore. Identifying Risk Factors for Webserver Compromise. In *Proceedings of the Eighteenth International Conference on Financial Cryptography and Data Security*, FC' 14, 2014.

[204] J. Vijayan. 'Hacker Safe' seal: Web site shield, or target? `http://www.computerworld.com/s/article/9057878/_Hacker_Safe_seal_Web_site_shield_or_target_?`

[205] K. Vikram, A. Prateek, and B. Livshits. Ripley: Automatically securing distributed Web applications through replicated execution. In *Conference on Computer and Communications Security*, Oct. 2009.

[206] W3C. Content Security Policy 1.1. W3C Editor's Draft 13, `https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html`, November 2013.

[207] W3C, editor. *How do you want to pay?*, Mar. 2014.

[208] J. Weinberger, A. Barth, and D. Song. Towards client-side html security policies. In *USENIX Workshop on Hot Topics in Security*, HotSec'11, Berkeley, CA, USA, 2011. USENIX Association.

[209] R. Wenning, editor. *W3C Workshop on Privacy and User-Centric Controls*, Berlin, Germany, Nov. 2014. W3C. Workshop agenda available at: `http://www.w3.org/2014/privacyws/`.

[210] R. Wenning and V. Galindo. Draft hardware security (hasec) working group charter. Technical report, 09 2015.

[211] M. West. Securing the client side. *Online at https://mikewest.org/2013/02/securing-the-client-side-devoxx-2012*, 2012.

[212] M. West. Play safely in sandboxed iframes. 2013.

[213] M. West. Mixed Content. *W3C Working Draft*, Sept. 2014.

[214] M. West. Content Security Policy Pinning. W3C Working Draft, W3C, 2015. Work in progress. `http://www.w3.org/TR/2015/WD-csp-pinning-20150226/`.

[215] M. West. Upgrade Insecure Requests. W3C Working Draft, W3C, Apr. 2015. Work in progress. `http://www.w3.org/TR/2015/WD-upgrade-insecure-requests-20150424/`.

[216] M. West, A. Barth, and D. Veditz. Content Security Policy Level 2. W3C Working Draft, W3C, Feb. 2015. Work in progress. `http://www.w3.org/TR/2014/WD-CSP2-20140703/`.

[217] M. West, A. Barth, and D. Veditz. Content Security Policy Level 2. W3C Working Draft, W3C, July 2015. Work in progress. `http://www.w3.org/TR/2015/CR-CSP2-20150721/`.

[218] WhiteHat. Website Security Statistics Report. `https://www.whitehatsec.com/resource/stats.html`.

[219] Z. Whittaker. Ubuntu forums hacked; 1.82m logins, email addresses stolen, July 2013.

[220] D. Wichers. Owasp top 10. *Online at https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project*, 2013.

[221] C. Yue and H. Wang. Characterizing insecure javascript practices on the web. In *International Conference on the World Wide Web (WWW)*, WWW '09, pages 961–970, New York, NY, USA, 2009. ACM.

[222] M. Zalewski. Postcards from the post-XSS world. [online], `http://lcamtuf.coredump.cx/postxss/`, December 2011.

[223] W. Zeller and E. W. Felten. Cross-site request forgeries: Exploitation and prevention. Technical report, Princeton University, 2008.